MISSION OPERATIONS AND DATA SYSTEMS DIRECTORATE

Integrated Monitoring, Analysis, and Control COTS System (IMACCS) Engineering Report

January 1996

National Aeronautics and Space Administration

Goddard Space Flight Center
Greenbelt, Maryland

Integrated Monitoring, Analysis, and Control COTS System (IMACCS) Engineering Report

January 1996

Preface

This report summarizes the results of the Integrated Monitoring, Analysis, and Control COTS System (IMACCS) prototype. Questions concerning this document and proposed changes should be addressed to:

Gary Meyers
Code 550
Goddard Space Flight Center
Greenbelt, Maryland 20771

Abstract

This report describes a successfully functioning, commercial off-the-shelf (COTS)-based ground support system called the Integrated Monitoring, Analysis, and Control COTS System (IMACCS). IMACCS was implemented as a prototype by Goddard Space Flight Center's Mission Operations and Data Systems Directorate (MO&DSD) to operate NASA's Solar Anomalous and Magnetospheric Particle Explorer. IMACCS was conceived specifically to build on previous experience in testbed evaluation of COTS products. The IMACCS project was to integrate a typical set of such tools, connect them to live tracking and telemetry data from a real on-orbit satellite, and perform shadow mission operations. The IMACCS project was to assess the completeness, robustness, and performance of a COTS-based ground system. As an additional constraint, IMACCS had to be implemented within 90 days of project approval. This report discusses the challenges that led to the IMACCS project, the processes used for implementing IMACCS, how these processes fit within MO&DSD's reengineered ground systems development processes, and the results obtained by comparing IMACCS requirements, operations, costs, and implementation process against the currently operating ground system.

Keywords: prototype, COTS, integration, ground data system

Table of Contents

List of Tables

List of Figures

Section 1. Introduction

The increasing availability, low cost, and automation potential of commercial off-the-shelf (COTS) hardware and software designed for spaceflight ground support make them an attractive prospect for substantial reductions in the cost of ground systems and flight operations. However, the prospect of using such tools, especially for scientific missions with complex operational profiles, is complicated by their relatively recent appearance—they have not been validated by extensive operational use. The Integrated Monitoring, Analysis, and Control COTS System (IMACCS) is a proof-of-concept demonstration of the ability of COTS products to meet the ground system needs of an existing mission at the National Aeronautics and Space Administration's (NASA's) Goddard Space Flight Center (GSFC).

Computer Sciences Corporation (CSC) and other organizations have conducted similar feasibility demonstrations using COTS and nondeveloped items (NDIs) or legacy software for developing ground systems. For example, the U.S. Air Force's Center for Research Support (CERES) COTS ground system project at the National Test Facility in Colorado Springs, Colorado, validated this approach for military spacecraft [Monfort, 1995]. In London, the International Maritime Satellite (Inmarsat) consortium also validated this approach for automating their operations. The CSC Integrated Ground Support System (CIGSS) and Eagle projects provided an environment to configure a ground-based data system to support spacecraft operations, using reusable application modules (RAMs), legacy software developed for GSFC, and COTS products [Werking, 1993]. These examples illustrate that interest in COTS-based ground systems extends throughout the spaceflight community.

The target spacecraft for this demonstration was the Solar Anomalous and Magnetospheric Particle Explorer (SAMPEX), which is the first Small Explorer (SMEX) satellite. This satellite is in a low-Earth orbit (altitude of 800 kilometers), with a payload of four scientific instruments. The mission has been flying without incident for 3 years and thus is an excellent candidate for this type of demonstration.

1.1 Objectives

The major objectives of this demonstration were to

- Compare IMACCS's performance to existing SAMPEX ground system performance
- Compare IMACCS's functions to existing SAMPEX ground system functions
- Compare IMACCS's user interface to existing SAMPEX ground system user interface
- Evaluate the ease and feasibility of integrating COTS products

Determine the lines of code that had to be developed to integrate the various products

Determine whether IMACCS could be operated by a single analyst or be unattended overnight

Determine the cost of developing and operating IMACCS compared to the existing SAMPEX ground system

By evaluating IMACCS against these objectives using live telemetry and tracking data, the extent of its operational viability can be determined.

The investigation was part of a larger NASA/GSFC effort, called the Reusable Network Architecture for Interoperable Space Science, Analysis, Navigation, and Control Environments (Renaissance) [Stottlemyer, 1993]. The purpose of Renaissance is to reengineer the process by which the Mission Operations and Data Systems Directorate (MO&DSD) at GSFC builds space communications, data, and information systems in support of flight projects. Renaissance plans to develop and operate ground systems more cost effectively, and with greater flexibility to meet individual customer needs, than has been possible in the past. These goals will be met with reusable hardware and software components to reduce schedule, complexity, and development cost (faster, better, cheaper).

To support this goal, a proposal was made to prove the concept and build a complete ground system entirely out of COTS components in 90 days. The 90-day timeframe was selected for the challenge because all parties felt that it could be done in 90 days and because it might be a future requirement. Future missions are expected to be planned and executed on short schedules. It was also recognized that the system would have to meet the operational requirements not only of SAMPEX, but also those of the coming generation of satellites—such as the Medium Explorer (MIDEX) and SMEX-Lite series of satellites—to have the greatest impact.

Therefore, the system developed had to be capable of operating, for the most part, autonomously, particularly in the area of command and control. Several COTS products are capable of meeting this requirement, including the following:

RTWorks, a rule-based expert system shell from Talarian that is not particularly oriented toward space missions

G2, a rule-based expert system shell, from Gensym

Intelligent Mission Toolkit, a space mission product based on G2 from Storm Integration, Inc.

Altair’s Mission Control System (MCS), a space mission product based on RTWorks from Altair Aerospace

Epoch 2000 from Integral Systems

OS/COMET from Software Technologies

The task was to assemble a ground system from representative products based on experience observing and using these products on other projects. Because it also was recognized early that legacy (NDI) products were necessary for the IMACCS effort, an additional objective was established that would measure the ease of integrating COTS with NDI products.

1.2 The Challenge

Traditional ground data systems (GDSs) at GSFC have been largely custom solutions built to satisfy a unique set of mission requirements. Budgets for these projects typically run in the $10 million to $20 million range, with 3 to 5 years to develop an operational system. Due to shrinking NASA budgets and changing customer wants and needs, this trend has been changing over recent years. In addition, MO&DSD, which has been the primary provider of GDSs to customers, is encountering stiff competition from other NASA elements, universities, and industry, which are now capable of providing suitable products to traditional MO&DSD customers. To meet these new market pressures, the MO&DSD director challenged the directorate to develop a GDS for under $5 million and in under 1 year.

To meet this challenge, Renaissance was established in response to an architecture study by GSFC. The first-generation Renaissance architecture was based on legacy software reuse. Subsequently, a similar approach was used for the Advanced Composition Explorer mission. These were steps in the right direction, but efforts still fell short of the goal. Concurrently, the CIGSS and Eagle projects provided an environment to configure a ground-based data system to support spacecraft operations using RAMs, legacy software developed for GSFC, and COTS products [Werking, 1993]. Still, a bolder approach was needed. Small groups at GSFC and CSC, familiar with and successful at implementing systems using an iterative prototyping process, were advocating empowerment of small teams to implement GDSs using a skunkworks approach. This approach addressed trends that were impeding rapid GDS development at GSFC. These are the drivers that led to IMACCS.

IMACCS is a proof-of-concept demonstration of the ability of COTS products to meet the challenge. An IMACCS prototype was produced using an iterative prototyping process to meet the GDS needs of an existing mission at NASA/GSFC.

1.3 Document Organization

Section 1 describes the overall objectives of the IMACCS project and the drivers or challenges that led to starting the project. Section 2 describes the COTS products that were used in IMACCS, the overall architecture that these COTS products were used to implement, and the hardware configuration. Section 3 describes the process used in implementing IMACCS.

Section 4 compares IMACCS to the currently operating SAMPEX legacy system by comparing requirements and operations. Section 5 describes IMACCS costs and how they compare to the current SAMPEX system. Section 6 describes lessons learned, and Section 7 contains evaluations of the COTS products used in IMACCS. The appendix presents a requirements comparison in more detail than what is presented in Section 3.

Section 2. System Description

2.1 COTS Product Selection

The first step in using COTS for GDS development is to select the COTS product suite by evaluating each COTS product against mission requirements. This step was not done for the IMACCS prototype. Tools chosen for IMACCS were representative in their capabilities and were chosen from among the available products based on a variety of motives. In some cases, the choice was made based on the willingness of the vendor to make demonstration licenses available. In other cases, some familiarity with the product existed within MO&DSD. For example, Altair's MCS was selected based on prior experience using this product on another CSC project.

The primary focus for this investigation was integration feasibility, not choosing an optimum tool set. However, the current maturity and applicability of the COTS products in IMACCS for use in the NASA operational environment was evaluated. Table 2–1 lists the COTS components integrated into IMACCS and the IMACCS function assigned to each COTS component. Figure 2–1 illustrates COTS/NDI products and their functions within the IMACCS architecture.

In addition to these COTS products, IMACCS interfaced to the existing Command Management System (CMS) using File Transfer Protocol (FTP) through an Ethernet local area network (LAN). CMS performs the command load generation function for SAMPEX. The command load processor, developed as glueware, converted SAMPEX command strings into proper Consultative Committee for Space Data Systems (CCSDS), NASA Communications (Nascom), and mission-specific formats for transmission. For IMACCS, glueware is defined as software developed using a high-level programming language (e.g., C or C++) or a scripting language (e.g., Perl). Because a SAMPEX ground system was being reproduced, commands had to be transmitted that SAMPEX could understand.

Level-zero processing (LZP), science data distribution, and science planning were not implemented in the prototype, but are part of the currently operating SAMPEX ground system. Orbit maneuver planning and gyro calibration are functions typically found in a ground system, but were not required for SAMPEX. However, COTS products are available that perform these functions. CSC's Navigator can perform orbit maneuver planning and The Math Works' MatLab can perform gyro calibration.

All COTS products were hosted on RISC-based workstations [Hewlett-Packard (HP) and Sun processors] under UNIX, configured on a LAN.

2.2 Architecture

The system architecture for IMACCS (Figure 2–1) consists of a loosely coupled set of tools that start with the communications front-end processor (FEP)—the Loral Test and Integration System Model 500 (LTIS 550). The front end receives telemetry (CCSDS packet telemetry) from the SAMPEX spacecraft and tracking data from the ground stations through the Nascom ground network. Likewise, through Nascom, the front end transmits command data from the command load processor to the SAMPEX spacecraft via the ground stations and transmits station acquisition data [improved interrange vectors (IIRVs)] to the ground stations.

Table 2–1. COTS Components and Their Functions

LTIS 550 FEP
Receive telemetry and tracking Nascom blocks
Synchronize CCSDS telemetry
Extract SAMPEX telemetry values
Convert to engineering units
Transmit command Nascom blocks
Transmit acquisition data Nascom blocks
Altair's MCS
Monitor and display data
Monitor subsystems with state modeling (for automation)
Prepare real-time commands
Analytical Graphics' Satellite Tool Kit (STK)®
Predict orbit
Predict events
BBN's Probe
Trend data
Manipulate and display data
Storm Integration's Precision Orbit Determination System (PODS)
Determine orbit using batch least squares
CSC's Raw Data Processor (RDProc)
Reformat tracking data for PODS
Time-order tracking data and remove duplicates
The Math Works' MatLab®
Determine attitude
Compute acquisition data

Talarian's RTWorks®
Provide data server
Archive and playback data
Provide display builder

In addition to receiving data, the front end also decommutates and converts the data to engineering units for other ground elements. Altair's MCS monitors spacecraft health and status, MatLab computes spacecraft attitude, and Probe analyzes and reports data trends using this data. Tracking data is collected by the front end, which is processed by RDProc for PODS and Chains, which determines spacecraft orbit. Although not shown in the figure, the STK Visualization Option (VO) was used to graphically display the satellite in orbit using a standalone silicon graphics processor. Incorporating many of the other functions of STK, VO allows the user to model both the orbit and orientation of the spacecraft and to animate ground contacts. Health and status monitoring and commanding are real-time functions. All others are offline functions.

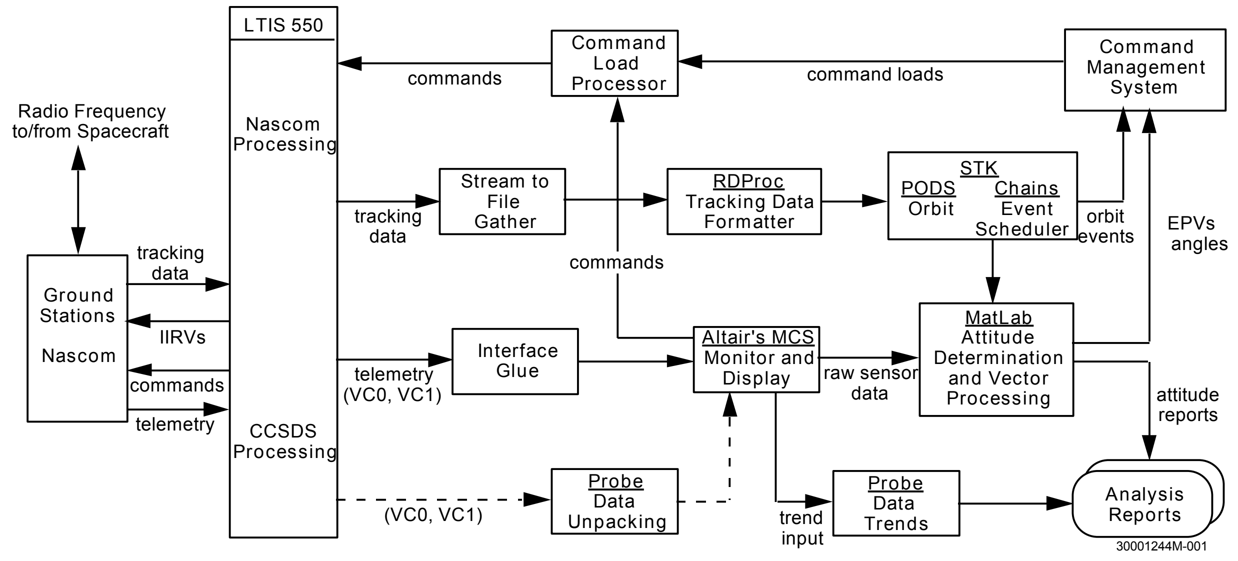

Figure 2–1. IMACCS

2.3 Hardware Configuration

Figure 2–2 illustrates the IMACCS hardware configuration. For final integration and demonstrations, IMACCS was distributed across different facilities. The LTIS Versa-Module European (VME) hardware was located near the Nascom Data Distribution Facility (DDF) and controlled by a Sun workstation located in the System Engineering Facility (SEF) in another building. The data was then placed on a GSFC campus Ethernet LAN for transfer to the SEF. IMACCS used the LTIS 550 communications FEP as its interface between the HP workstations and the Nascom serial communication lines. In addition, the CMS, located in the SAMPEX Mission Operations Center, also interfaced with IMACCS via the same LAN.

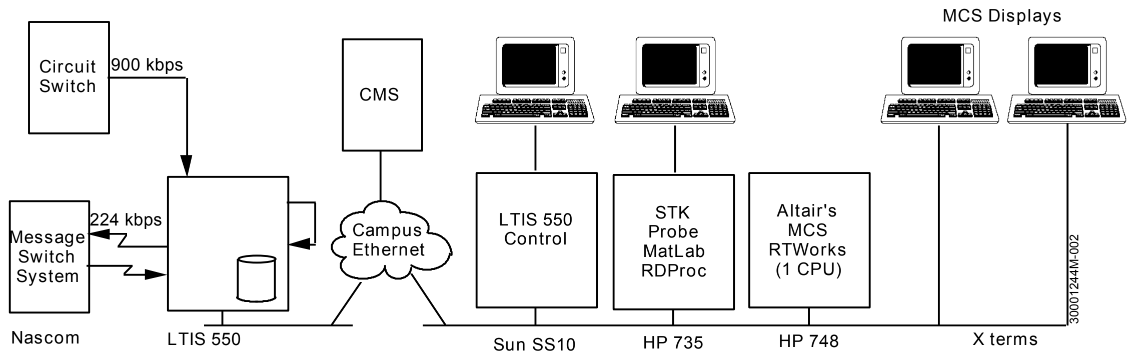

Figure 2–2. IMACCS Hardware Configuration

Section 3. IMACCS System Implementation Process

3.1 Process Overview

The Renaissance project has proposed [NASA/GSFC, 1995] a process that, if adopted, would enable MO&DSD to better meet customer needs using a customer-focused, intensive teaming approach. Advances in technology (e.g., online work group systems and computer repositories), the need for fewer developers, and the availability of off-the-shelf products has made this approach possible. Primarily, off-the-shelf products are integrated using an iterative implementation process throughout the mission life cycle.

To exercise and evaluate this process, the IMACCS team attempted to build a complete ground system entirely out of COTS components in 90 days. Much of the prototype process was concerned with configuring, integrating, and verifying products. The iterative prototype process used in IMACCS included an integration strategy where, as the pieces of the system were completed, they were integrated and tested. Feedback was obtained through prototype demonstrations.

3.2 Product Configuration

Having determined which COTS products to use prior to the start of the 90-day period, the next step in prototype implementation was to acquire the COTS products selected. Because IMACCS was primarily a prototype demonstration and not expected to be used operationally, no products were purchased. Vendors were contacted and, in most cases, provided demonstration copies of their products for the 90-day integration period. Once secured, the products were installed on workstations in the Renaissance SEF at GSFC. Altair's MCS had not been hosted on HP workstations prior to the IMACCS project. Therefore, a workstation was moved from the SEF to the Altair facility to enable Altair to perform the port. It should be pointed out that at the beginning of the 90-day period, the only pertinent hardware and software that existed within the SEF were the HP workstations and their operating systems. The remainder of the COTS products were requested after the start of the 90 days.

Before it could be used, each product had to be configured to operate within the target environment. In many cases, this involved building system tables to assign values to parameters. Both the LTIS 550 and the MCS are designed to incorporate a database of telemetry and command descriptions. Both applications needed to be configured to recognize the format of the SAMPEX project database. Additionally, the project database itself needed to be consolidated from several files into one.

Other aspects of IMACCS required a much larger configuration effort. For example, to identify and respond to spacecraft conditions in the MCS, state definitions and contingency procedures are used and must be configured ahead of time. A state definition is a specific telemetry signature or set of telemetry points that indicate that a particular condition exists [Wheal, 1993]. States may be defined for all known or expected conditions that can be identified by a particular telemetry signature. When an undesirable state is encountered, contingency procedures can be executed to transition the spacecraft to a desired state. Not all SAMPEX states were defined for the prototype. In addition, displays were built that show an operator the current condition of the spacecraft as derived from these state definitions.

Likewise in the front end, the cascading processes that take the data through Nascom and CCSDS translation, and ultimately populate a table of current values of all telemetry, required a great deal of configuration and reconfiguration. During this integration, one of the most valuable aspects of using COTS for system development was realized. During regular team meetings, as the subsystem leads shared their progress, it was discovered that the trending tool, BBN's Probe, could be used as an integration aid because of its robust data interface. This experience points out the value of COTS features and their unexpected applicability.

3.3 System Integration

The next major effort was integrating the various elements. This effort involved defining the interfaces and developing the necessary glueware to implement each interface. Glueware was developed to convert commands and command loads into a form suitable for transmission by the front end (the command load processor), to enable the LTIS 550 to gather tracking data, and to enable the MCS to receive telemetry data from the LTIS 550. In addition to glueware, a scripting language (Perl) was used to create code that enabled the LTIS 550 to extract telemetry parameters from telemetry.

Several significant obstacles were encountered in the configuration, installation, and integration effort:

Difficulty in performing the installation (direct assistance from the vendor often was required)

Difficulty in getting products to interface with one another as initially expected or advertised

Difficulty in getting the product to work as advertised

Expiring licenses

Hardware failures

Difficulty in getting demonstration versions of a COTS product to do all the functions needed

Software bugs

All obstacles were remedied within the 90-day integration period by working with the product vendors, who were extremely helpful and supportive, or by implementing a solution using an alternative from within the selected COTS products.

3.4 Final Evaluation

Successful integration was defined as complete end-to-end throughput with telemetry received and processed and commands generated. Table 3–1 illustrates when the most important events occurred during the 90 days. At the end of the 90-day period, IMACCS could read and display all telemetry and tracking data, process this data into attitude and orbit solutions, trend the data, display orbital events, and perform station acquisition data computation. The commanding capability was not complete however. To allow for SAMPEX-specific command formats, a mission-specific reformatting process was needed; custom glueware had to be written (the command load processor). The commanding capability became operational on the ninety-fifth day.

Table 3–1. Important Milestones

Date (1995)	Milestone
Apr 17	Day 1 of 90
May 15	LTIS 550 arrived
Jun 4	LTIS 550 connected to Nascom circuits
Jun 8	Nascom data deblocked by LTIS 550
Jun 20	72-byte tracking data extracted with LTIS 550
Jun 30	IIRVs (acquisition data) and extended precision vector (EPV) (onboard ephemeris updates) produced in correct form
Jul 5	Telemetry received and processed through engineering unit conversion; validated orbit determination results
Jul 14	Altair's MCS displayed all telemetry points in graphical and alphanumeric form; attitude control subsystem and electrical power subsystem state model functions implemented
Jul 16	Day 90 of 90
Jul 18	Operator command capability implemented
Jul 21	Command load capability implemented
Jul 28	Successfully commanded the SAMPEX engineering test unit (ETU)
Oct 11	Successfully commanded the SAMPEX spacecraft

A major goal (although not a requirement for declaring the prototype a success) at the start of this project was to have the system command the SAMPEX spacecraft. Prior to performing this function, however, tests had to be performed to validate the fidelity and reliability of the command function. Operational scenarios were selected with the support of the Flight Operations Team (FOT). The system generated commands according to these scenarios.

To qualify the IMACCS commanding capability, an event log and dump of all commands executed was acquired from the SAMPEX FOT. This data included the sequence of commands that loads the stored commands into the onboard computer. Using IMACCS, the same commands in the SAMPEX Software Development and Validation Facility were executed using the ETU and then dumps of commands executed were captured. The two sets of dumps were compared byte by byte to ensure that they were equivalent. IMACCS successfully commanded SAMPEX on October 11, 1995, using a Madrid (Spain) pass.

For trend analysis and flight dynamics products, computations were compared with operational products that are produced on a regular basis for SAMPEX.

Section 4. IMACCS Compared to Legacy System

4.1 Requirements Comparison

The original objectives for IMACCS were to meet 80 percent of SAMPEX's functional requirements at approximately 10 percent of the cost of developing custom ground support software. Table 4–1 shows the number of SAMPEX flight dynamics, control center, and CMS requirements that were and were not satisfied by IMACCS. As shown, IMACCS satisfied 109 of the 125 relevant SAMPEX requirements and did not satisfy 16 SAMPEX requirements in these categories. Therefore, IMACCS met 87 percent of the relevant SAMPEX requirements. IMACCS met all SAMPEX functional requirements for command and control, including the front-end and Nascom interface, orbit and attitude determination, data trending, CMS, and mission planning. A detailed table of the SAMPEX requirements is presented in the appendix**.** All telemetry parameters were processed through engineering unit conversion, with stripchart and alphanumeric display generation for all these quantities possible. IMACCS also was capable of operator command and command load uplink.

In addition, the COTS products used in IMACCS provided several additional features not found in the legacy system. These features are particularly beneficial because they offer added flexibility and the capability to reduce the number of operators needed. They include

Dynamic display modification

Arithmetic and statistical manipulation of trended data

State space modeling—increased levels of automation capable with intelligent monitoring

Health and safety monitoring provides the greatest promise for increased operations automation. This potential was not fully realized for IMACCS due to the 90-day time constraint.

4.2 Operational Comparison

Based on demonstrations seen, the FOT believes that many added benefits exist with IMACCS when compared against the current operational system. State modeling significantly eases anomaly resolution and enables the FOT to easily maintain the models (with training) to reflect their experiences. There is a significant potential for increasing automation, and the open systems architecture, using COTS products, provides an environment to easily adapt and optimize the system to suit FOT needs. Some of the major shortcomings expressed by the FOT deal with features not fully implemented in the prototype, but that would most likely be provided if an operational version were developed. For example, not all state models were developed, and data accountability parameters from the front end were not fully implemented.

Table 4–1. Functional Requirements Satisfied with IMACCS

Requirement Category	Satisfied	Not Satisfied
Flight Dynamics Requirements		
Orbit Determination	4	0
Attitude Determination	3	0
Mission Analysis	0	0
Planning Aids	2	0
Special	3	1
Flight Dynamics Total	12	1
Control Center Requirements		
Functional Interfaces	8	1
Real-Time Telemetry	12	1
Special Real-Time Telemetry	6	1
Real-Time Telemetry Report	6	4
Playback	4	1
Commanding	11	0
Display	12	0
User Procedures	3	1
History	7	1
Database	1	1
Scheduling	0	4
Control Center Total	70	15
CMS Requirements		
File Setup	1	0
Load Generation	6	0
Pass Plan	1	0
Table Maintenance	1	0
Uplink Table	5	0
Event Log	1	0
Display	1	0
Reports	6	0
Performance	5	0
CMS Total	27	0
Grand Total	109	16

Section 5. Cost Analysis

IMACCS costs can be divided into hardware costs, software license costs, and labor costs. Each cost category is further divided into the cost of procurement or production of the system and the cost of maintaining the system. Hardware costs are estimated for one front end, two workstations, two X-terminals, and networking hardware. Estimated software license costs cover the license prices for one seat of each tool. (STK VO is not included in this cost estimate.) Labor costs are estimated for system integration and configuration. This effort does not include requirements development or product evaluation.

Table 5–1 shows cost estimates for the prototype and for a robust, operational system built with IMACCS components. The IMACCS team includes civil servant, contractor, and subcontractor labor hours.

Table 5–1. Cost To Develop IMACCS

Costs	IMACCS	Operationa l
Hardware	$250,000	$450,000
Software license	$200,000	$450,000
Labor	$125,000	$300,000
Total	$575,000	$1,200,000

These totals compare favorably to the current ground system target of $1 million to $2 million.

Yearly software maintenance fees are 18 percent of the license costs ($27,000 to $81,000), and annual hardware maintenance is estimated to be $10,000. An assessment is being made as to whether the original cost objective (10 percent of a custom-built system) has been met.

Glueware is defined as coding that had to be written using a conventional programming language, such as C or C++, or a scripting language, such as Perl, to interface products together or to implement a function not performed by a COTS product. Most of these cost estimates are presented in Section 7. However, for the command load processor, approximately 1500 lines of C code were written as glueware to perform real-time commanding. Most of this code was taken and reused from another mission.

Section 6. Lessons Learned

COTS products provide an unprecedented capability to build a ground system faster, better, and cheaper than using more traditional development approaches. Even though this demonstration is only one example of using this capability to build a ground system, the lessons described in this section may be useful to others conducting similar projects in the future.

6.1 Process

The IMACCS project was modeled on the concept of skunkworks, which was assumed to mean working toward a goal in the most efficient manner discovered. Team members, some volunteers, and some personnel assigned by management were brought together and given the elements of the project. These elements included a clear statement of the end product (i.e., what the prototype would do), constraints, and which resources could be drawn on. Each member of the team had some experience that could be brought to bear on the problem, and the team was encouraged to communicate freely to take the best advantage of everyone's talents. The process included seven aspects: product selection, procurement, familiarization, configuration, integration, test and checkout, and demonstration.

6.1.1   Product Selection

What was done: In effect, the product selection process was separate from the IMACCS effort. The decision to begin the prototype was preceded by selecting the COTS packages that would be used. Most packages were selected based on experience that either CSC or GSFC had had in previous applications. Thus, BBN's Probe and Analytical Graphics' STK were in use (although not in support of missions) in the Flight Dynamics Division (Code 550). Altair's MCS was selected because CSC had worked with Altair Aerospace on some joint demonstrations and recommended putting it to more rigorous scrutiny in an operational context. In the case of selecting a front end, while none of the participants had any prior experience with COTS front ends, Loral AeroSys used its organizational connections to borrow the LTIS 550.

What worked: The products used were well-suited to the application. In several cases, reliance on technical support from the vendors was required, which they all supplied quite generously. In all cases, the packages were adequate for the IMACCS application.

What did not work: Because the team did not go through the selection process, there was a lack of familiarity with the products themselves. This drawback resulted in more time being spent learning to run the applications than would otherwise have been necessary.

Recommendations: It is extremely important to evaluate each product thoroughly to make an informed choice. As indicated in Table 6–1, the evaluation criteria include functional sufficiency as well as things that make a product easy to integrate. The primary focus of the evaluation, which unquestionably requires running the software and exercising its features, is to determine whether or not it does the job needed. This includes accuracy as well as just being able to produce the output.

__Table 6–1. COTS Product Evaluation Criteria__

Evaluation Criteria
Does it work?
Does it completely satisfy a particular function, or does it need to be augmented with another product?
Availability of hands-on time – Load testing – Throughput – Environment setup compatibility – Platform/operating system compatibility
Same version and use of ancillary packages, e.g., database management system (DBMS)
Similarity of user interface (to some ideal)
Availability through scientific and engineering workstation procurement
Availability of (good) documentation
Vendor interaction – Good personal contact – Responsive technical support – Accurate information
Automatability – Scripting access and capability
Configuration flexibility
Licensing arrangement
Error messages – Comprehensive – Comprehensible – Useful

However, there are other important factors too. For example, if two packages are using a database application, can they use the same one? Must they use the same one? If they use the same one, do they have a conflict in the database structure? Do different packages require communications ports to be set up differently? Is the package compatible with the platform and operating system on which it will be installed? Is the user interface similar to others in the system? Does the vendor offer good technical support?

The best products for this type of effort will come from vendors who are willing and able to support clients on their terms and schedules (clients cannot deal with technical support people sending software problem reports to a separate engineering group for inclusion in some later version of their product).

There are some specific attributes to avoid, as well. Do not buy version 1.0 if it can be avoided. If there is a choice, do not buy a product for which an evaluation copy is not available.

6.1.2 Procurement

What was done: For IMACCS, as opposed to an operational system development, procurement was the process of cajoling vendors into making their products available for free. This is actually the appropriate course of action involved in product evaluation (part of the selection process). In any case, whether borrowed or bought, the packages need to be received and installed. The IMACCS team had a single point of contact for procurement. This person telephoned all the individual vendors, explained the nature of the project, provided the pertinent information about the destination environment [e.g., operating system, medium, Internet Protocol (IP) address], and kept the correspondence alive until the package was delivered.

What worked: In the first few weeks, procurement was essentially a full-time job. It was good to have a single person keeping track of where all the negotiations stood.

What did not work: For one vendor, Altair, engineering support was contracted for assistance in configuring the MCS. CSC's Purchasing Department was most helpful in the process, but it is expected that the process will be more difficult in the future. The process took 7 weeks, which is too long considering that the time allocated for the entire project was 12 weeks. The procurement process took more than half the total time allocated to develop a prototype IMACCS. A future team should try a completely different process for purchasing vendor support—it could only be better.

Recommendations: Except for the difficulty with purchasing vendor support, the experience was a positive one. People should be prepared for the process to take longer than it does to buy personal computer (PC)-type software; the sophisticated software involved in GDS integration requires more cooperation between buyer and vendor.

6.1.3 Familiarization

What was done: Rather than spend the time becoming familiar with all the features and quirks of the various products, the IMACCS team concentrated on using the various COTS packages to perform GDS functions. Thus, the team never asked or answered the question, "What are all the things Product X can do?" Rather, one team member was given the task of using BBN's Probe to perform trending or using STK to produce acquisition data.

One of the first steps usually involved several days of going through the tutorials and users guides. As the individuals became more familiar with the environment (many IMACCS team members had had no prior experience in UNIX networking), they began to recognize similarities in user interface and configuration options. As described in Section 6.2.2, the team met regularly to discuss status and share technical information. In these meetings, team members were prolific in their ability to share shortcuts and devices among diverse applications.

What worked: Although it was not planned this way, the team member responsible for any given function was not particularly expert in that field. This turned out to have a significant benefit because there was no tendency at all to adhere to any extraneous procedure or method. In this manner, the most straightforward way of implementing the function was found. For example, in using STK to produce station access time predictions, there was no compelling reason to reproduce the batch capability of the Flight Dynamics Facility's (FDF's) PSATGen, which would have been impractical with STK.

What did not work: While it may just be a question of running out of time more than a failure of the process, the team was considering two additional COTS packages that it was unable to work into the prototype: BBN's Patterns and STK's Generic Resource, Event, and Activity Scheduler (GREAS). Although complete functionality was possible without them, each of these packages has some attractive features that would certainly have been exploitable if the team had had more familiarity with them.

Recommendations: Product familiarity is a skill that needs to be encouraged. Any additional time that can be allocated to a project can be used by individuals to familiarize themselves with the packages to be used in the system. A great deal of time today is spent coding and unit testing; some of the time saved by using COTS should be invested in familiarizing people with available packages.

6.1.4 Configuration

What was done: The IMACCS project was undertaken in the Renaissance SEF. The IMACCS prototype was imposed over an existing network configuration. As each new package was delivered, installation was handled however the vendor-supplied installation procedure dictated. Any scripts and data files generated by the team were kept in a single directory (once they were operational) on the network file system, but those supplied by vendors were left in the application directory, usually on a single host file system.

What worked: The UNIX operating system supports the scattered (or distributed) configuration that was achieved. Had it not, something else would have been used. A key component of configuration management is a regimen of regular system backups. One of the five workstations (a sixth was relocated to the Altair facility in Bowie, Maryland, for the duration of the project) was backed up to tape each day; an entire cycle of backups was completed each week. This proved crucial when key files included in the command load processor mysteriously disappeared. Had the files not been recovered, not only would several days' work have been lost, but it might not have been possible to reproduce the thought processes that had gone into them. Fortunately, a recent backup existed, which enabled the files to be restored.

What did not work: With different individuals and sub-teams making rapid and frequent changes to configuration files, it is easy to fall into a dangerously chaotic state. Someone with both a clear understanding of the end product and sympathy for the exigencies of the process needs to be the ultimate arbiter of changes to the configuration. In this case, the key command and monitoring screens were in flux right up to the very end. Vendor assistance (Altair Aerospace in this case) was needed and readily forthcoming. When the vendor made changes remotely, however, problems occurred in the compilation of scripts for executing the demonstration scenario. The problem was resolved by stricter management of the configuration by the team leader. After the first discovery that something that the team knew how to do yesterday was no longer possible today, every change to the operational libraries was required to be authorized by a single person—the team leader. Any changes that could possibly impact the demonstration were scrutinized very carefully.

Recommendations: It seems plausible that a better configuration (from the viewpoint of maintenance) could be achieved if more thought had gone into it up front. It might have been better to try separating the configuration into scripts, data, programs, etc., with everything that could be moved out to the network file system put there. This is a problem that UNIX systems seem to cultivate, but it would be good to control it better. A central person or group should coordinate all configuration activities from the onset of the process. However, this configuration management should be loose enough at first to accommodate flexibility. By the time interfaces are being established between functions, the configuration should be controlled closely (but not frozen).

6.1.5 Integration

What was done: Although an integration plan did exist, the various functions were not ready for integration as scheduled. The first fully integrated function was orbit determination because it had the fewest nodes along the way. Tracking data came in through the front end, was picked up by RDProc, and was passed into PODS. In contrast, while health and safety monitoring involved only two COTS packages (the front end and the MCS) internally, there were a great many nodes where data diverged, changed, and otherwise became a tunable parameter between the processes. During the process of integrating the front end with the MCS, it was discovered that BBN's Probe could be used quite effectively as both an integration tool and a trending tool (sort of a virtual oscilloscope). Thus, the data could be stripped out of packets from transfer frames captured by the front end and passed to functions downstream from the MCS to expedite their integration before the MCS could be made to serve the data.

What worked: Probe worked. Patience worked. Scripts written in Perl were invaluable for generating large interface configuration files from the project database.

What did not work: The team was unsuccessful, in the time it had, in serving the data in both packet and data-value form. This is a shortcoming that needs to be corrected.

Recommendations: As different pieces are ready to be integrated, it is possible that the integration of one function will undo an important feature of a different function. System integration must be rigorously controlled, either by the project leader or someone else with a broad perspective. If possible, a single individual should be responsible for all files. This action will prevent files from being overwritten when multiple functions use the same file.

6.1.6 Test and Checkout

What was done: Because the SAMPEX spacecraft is on orbit and functional, each IMACCS function was validated against its operational counterpart. Thus, the definitive and predictive orbit, as generated with STK and PODS was compared to the FDF product of the same time. Likewise, selected data points as monitored in the MCS were compared to reports from the operational Transportable Payload Operations Control Center (TPOCC). By the same token, IMACCS produced hex dumps of the command output and compared them to hex dumps of the TPOCC commands. Once satisfied on a piece-wise level, the team scheduled time with the spacecraft simulator (the ETU) and tested in closed loop mode. As a final test, live commands were sent to the satellite on October 11, 1995.

What worked: Apart from the accuracy of the numbers and messages, the team checked out the look and feel of IMACCS by inviting and encouraging the SAMPEX FOT to try it out and say whether or not the system had all the right gadgets. In most cases it did, but the team discovered in this fashion that it differed in interpretation of command sequence counter (which was easily and quickly corrected).

What did not work: Not applicable.

Recommendations: There is nothing peculiar to COTS integration about system level testing. It is an important part of any system development process. Unit-level testing is presumed done by the individual vendors and should be confirmed, to the extent of interacting with the system environment, by the system administrator.

6.1.7 Demonstration

What was done: Three people were designated to conduct the demonstration, but after a few months two were adequate for most audiences. The Government point of contact introduced the demonstration by giving the background and describing the GDS architecture. The demonstration was divided between the real-time and offline functions, and the real-time functions were demonstrated in both traditional and enhanced operations concept. The team had an outline to follow and a story to tell.

What worked: Practice made, if not perfect, then at least pretty good. Although it was tiring, the team found that having the same group conduct all the demonstrations developed confidence and resonance. The outline added structure and allowed the team to remember which points it was trying to get across.

What did not work: The first couple of demonstrations were done ad lib. Team members thought that because they were so familiar with the system and its history, they did not need any coordination or notes. Not true.

Recommendations: Do not underestimate the importance of a good demonstration, especially in a prototype project. People need to be able to really appreciate (i.e., see and understand) the data flow and the process. They cannot see it if it is not pointed out to them, and they cannot understand it if it is not placed properly in context.

6.2 Infrastructure

In addition to the process followed, the infrastructure in which the team operated was critical to the success of IMACCS. This includes management and communication.

6.2.1 Management

What was done: The overall theme was empowerment. The IMACCS team was lead by one civil servant (who was the primary civil servant on the project; another civil servant helped with procurement and configuration of some of the software) and one contractor. These individuals assembled the team and assigned responsibilities. They established meeting, documentation, and communication schedules. The contractor leader was the sole point of contact with contractor management and reported progress to them on a regular basis. The technical team was expected to act as dedicated and conscientious professionals; this they did without exception.

The manner in which problems were tracked and resolved was dictated to a large extent by the short timeframe. Each team member resolved his or her own problems as they arose. The team leader kept track of when various functions were expected to come online. The resolution of any problem that interfered with that schedule was elevated to the project To-Do List kept by the team leader and reviewed at the weekly meeting. In effect, problems were resolved as they arose; tracking them was a matter of days at most.

What worked: IMACCS benefited from a judicious application of management by advice and suggestion, not status monitoring. For example, when the schedule was tight, all of the technical people knew it. It would have been gratuitous for management to point it out.

What did not work: In the beginning, it seemed like a good idea to develop a precise schedule, accounting for every one of the precious 90 days allotted to the project. Unfortunately, the schedule was soon confounded by the erratic delivery of the COTS packages, and its maintenance was too time consuming to continue.

Recommendations: The IMACCS experience in this regard was thoroughly positive and ought to be used as a model for future projects. In particular, teams should be essentially rebadged so that other affiliations are temporarily forgotten. Managers should trust team members to get the job done; the team should be expected to provide frequent and substantial accounting of progress and problems.

The IMACCS team was empowered to complete an objective in a short period of time. The task was well defined, and the expected end result was well understood. Hence, the process used in IMACCS is suitable for similar kinds of projects.

6.2.2 Communication

What was done: A streamlined communications plan was prepared to keep management informed of progress. Part of the communication plan was a weekly progress report submitted to all Systems, Engineering, and Analysis Support (SEAS) managers interested in IMACCS. The IMACCS technical lead kept a log that recorded the technical status of the project. This log was used to generate the weekly progress report for managers. These reports kept managers informed of the current state of progress on IMACCS. Because the technical team was given maximum autonomy and authority to act, these reports enabled management to retain responsibility.

Intra-team communication was informal but sufficient. The team met each week to discuss the entire project. The meetings were scheduled to keep track of progress and to have a mechanism to communicate uniformly with the entire team at once. Because the team was small enough to accommodate at one time, team members could share ideas about what to do next, offer suggestions across the entire activity, and generally work together. In addition, all the work was done in a single facility. This contributed to communication between individuals almost continuously and kept everyone interested in and informed about the entire project.

What worked: One factor that contributed positively to intra-team communication, and which should be repeated, is to limit project size to 10 to 15 people, including civil servant and contractor leads. If more work must be done in less time, it is recommended that the core team be kept to 12 members and that additional personnel support this core team. A key feature of the approach used on IMACCS is that the different functional elements were not isolated from each other. This was only possible because the integration team was small enough to work closely and simultaneously.

What did not work: While the communication on this project was excellent, it could have benefited from slightly more formal intra-team communications, e.g., brief written progress reports from each member, emphasizing in particular any problems being encountered. It was discovered, for example, that MatLab could far more easily accomplish some tasks that were being attempted using STK. This could possibly have happened sooner. In addition, perhaps the technical lead's progress reports, or a modified version thereof, could be circulated among task members, giving them more awareness of the big picture. It should be emphasized, however, that the degree of independence allowed team members was outstanding and far outweighs these minor problems.

Recommendations: Communication between the core team and support personnel also should be considered. On IMACCS, communication between the core IMACCS group and the Altair Aerospace vendor team (who configured the basic state models and the interface to the front end) was facilitated by a core team member who was responsible for that function. The task was complicated, however, by the geographical separation of that work (performed at the Altair offices in Bowie, Maryland) from the rest of the IMACCS activities (at GSFC). Work should be done in a single facility where possible.

6.3 Miscellaneous Recommendations

6.3.1 Vendors

In general, most of the COTS vendors were eager to collaborate and generous with demonstration licenses and technical support. Some vendor firms were small and overextended, which made it difficult to obtain their support even though they were eager to provide it. Other vendor firms were large and bureaucratic, which often resulted in difficulties locating the right person to talk with to get a problem resolved expediently. Vendors usually were eager to assist in giving demonstrations.

Dealing with vendors is an important and large part of the integration process. Vendor negotiations and interaction were primarily the IMACCS lead engineer's responsibility, which was not necessarily the best approach. A distinction should be made between the licensing and logistical agreements that should be carried out by a single point of contact at the system level and the exchange of technical information that should be held at the subsystem (functional) level. This can sometimes be complicated if the vendor contact is the same in both cases. It is important to establish a personal relationship with an individual at the vendor organization. It is not important whether that person is the correct individual for every contact. An established person-to-person relationship will ensure that all issues receive appropriate attention and that communication is clear and consistent. Each party will know how to interpret the other's responses (whether assurances or instructions) and gain a sense that the matter will be resolved.

6.3.2 Analyze with Prototypes Instead of Paper

Prototyping forces people to understand and deal with all of the issues involved with passing data through all parts of the system. Projects such as IMACCS are important to developing viable and cost-effective solutions for the future. Managers should define creative solutions, use prototyping for proving and/or verifying the underlying concepts, and deemphasize paper analyses.

Working with the front end, the team found that memory consumed for packet extraction was excessive, but did not discover this until it had been tried. In response, the vendor implemented a solution they might not have been inclined to do without concrete demonstration. In general, issues of performance are unknown until the actual prototype is attempted.

In the orbit products area, the use of the STK Programmer's Library (PL) looked good on paper, but was much more cumbersome in practice. In the end, the team opted for an alternative implementation using MatLab.

A powerful result of actual prototyping is an understanding of the level of cooperation to expect from a vendor. In this regard, the IMACCS experience conformed to evidence from related but less ambitious prototypes.

6.3.3 Skills

COTS integration projects require a new set of skills in addition to those that have been required for traditional development.

Negotiation—Effective communication skills are essential and should be taught. Exercising sound judgment and good communication are the keys to successful negotiations. Generally speaking, vendors and integrators have similar and overlapping interests. Proper negotiations make that coincidence of interest obvious; poor negotiations disguise it.

Because IMACCS was a proof-of-concept project, the vendors selected had an opportunity to penetrate a new (and potentially large) market. The Renaissance budget did not allow purchase of any of the products. The goal of negotiations was to highlight the coincidence of these diverse interests.

Product familiarity—Integrators should become familiar with a product line so that the right choices can be made. Trade shows, conferences, user groups, and the Internet are all useful venues for this and should be encouraged by management.

Synthesis—Integrators must become familiar with integrating pieces to produce a solution. This can and should be treated as a valuable skill and encouraged by management. The best way to acquire this skill is by experience; the best way to garner the experience is by prototyping experiments.

Nimbleness—Rapid integration and prototyping involves coping with problems that emerge during integration, as well as pursuing alternative products in the background. People are needed who can learn new products quickly without forming too many emotional attachments to them. For example, when the team was delayed (and at the time it seemed that the delay might prove permanent) by difficulties in the CCSDS processor in the LTIS 550, team members were able to identify and exploit an alternative capability of BBN’s Probe to continue the integration of other functions. Likewise, the team changed the whole internal architecture it had presupposed for telemetry decommutation when the original idea proved to be too slow and too memory intensive.

6.3.4 Risks

Several risks existed in using the IMACCS approach to ground system implementation. A perceived risk was the feeling that COTS products lacked the functionality and robustness needed to implement a complete ground system. IMACCS proved this to be a fallacy. Another risk was vendor demise. Many COTS vendors are small, startup ventures that are at risk of going out of business. Although not done for IMACCS, source code can be held in escrow and is available to the purchaser if the vendor does go out of business. Another perceived risk was that the COTS and Government off-the-shelf (GOTS) products from different sources would be difficult to integrate. This was not at all the case in so far as making the products work together is concerned. A more difficult problem, and one that IMACCS did not address, was conforming all the products to the same user interface, giving them all the same look and feel. The IMACCS project did not attempt this, nor did it treat the problem as a high priority. The most significant risks found were the lack of a well-defined and documented process at project start and lack of the appropriate personnel skill mix for a COTS integration project.

The time and cost savings associated with COTS integration can offset risk. Projects should integrate their systems early, and thereby spend more time in spacecraft integration and test, reducing mission risks.

The chance that a product will not perform as advertised is not a risk—it is a certainty. It can be mitigated however by working closely with vendors to clearly understand the product beforehand. It can be further mitigated with other products. That is, what functionality any given product lacks or performs in an inadequate fashion, can be compensated by the integration of another COTS application that performs the missing function.

6.3.5 COTS Capabilities

The products varied in maturity, but were found to be capable of being applied to IMACCS. Documentation was uniformly difficult to use, and because vendors were geographically distributed, communications were somewhat hampered. Electronic mail partially overcame this difficulty. COTS products offer tradeoffs in flexibility and configuration ease. In the case of LTIS 550, it was useful for LTIS to telnet into IMACCS to check out problems.

Many vendors naturally are optimistic in their claims of system maturity and capability. In one case, the vendor actually built software almost interactively with the team. Also, many of the vendors' claims about functionality did not always work as expected. Some functions existed, but had never been used; some functions were designed into the product, but never implemented; and some functions that vendors claimed the product had did not exist. In this regard, the team received (and others ought to expect) support from the vendors to make the product perform as advertised. In general, the products that had large markets needed less alteration than products that were more tightly directed. In other cases, the product had unadvertised features. Also, tools could be combined in unexpected ways to provide capabilities not originally envisioned. Some of the features provided by the product enabled implementation of capabilities beyond those currently being used in operations.

System integrators need to remember that these COTS products contain many ways of getting data in one end and out the other; most of the effort is spent configuring the interfaces between products. Some products were somewhat unreliable because it was difficult to determine why a particular problem kept occurring. Other products were awkward to use and did not process data directly as desired, which necessitated that additional glueware be developed.

Another important lesson to remember when using COTS products is that licenses expire. This was particularly evident for the prototype because demonstration versions of the products were used, but it needs to be considered for any development effort. However, although these products were developed independently of each other, interfaces could easily be developed to produce an integrated system.

Many packages with overlapping capabilities are available in the marketplace. Many vendors claim that their single product is all that is needed to provide adequate mission support. To some extent, this claim is made by BBN's Probe, Altair's MCS, Analytical Graphics' STK (with VO), and probably LTIS, but it is never true. Choosing the most appropriate application for any needed function may still be more art than science. More instances will be needed before enough data will exist to generalize about the circumstances where one application is indicated over another.

Some features work on a given platform and not on another. More flexibility in the choice of platform (e.g., the SEF is essentially equipped with HP workstations only) would have been a benefit.

6.3.6 Use a Dedicated System Administrator

It is important to have a dedicated system administrator. On IMACCS, a skilled person provided administration service, although periodically system administrator attention was lacking. Future integration teams should try to maintain a full-time system administrator. If this is not possible, a full-time team member should be the part-time system administrator. However the job is allocated, a system administrator needs to be responsive and efficient, especially on a severely constrained project such as IMACCS. Time lost waiting for accounts to be set up or packages to be installed can be extremely detrimental.

Section 7. Specific COTS Product Evaluations

7.1 Altair’s Mission Control System

Function performed: Real-time commanding, health and safety monitoring, and data archival and retrieval

General impression: Strong tool for real-time operations, robust architecture, unique (expert system) capability

Good points: Easy to configure by operators and integrators; tools available to aid model and display generation and modification

Bad points: Limited interface options for integrating telemetry data into the system; little documentation on concurrent processes and what their interplay is

Integration effort: The spacecraft command and telemetry databases had to be analyzed to determine bindings for the MCS software bus. The data type (i.e., numeric or string) for each telemetry mnemonic had to be determined, a data group had to be assigned (virtual communications channel), and input files for the MCS variable-table-build process had to be generated.

Another significant effort involved gaining an understanding of the SAMPEX attitude control subsystem and electrical power subsystem. State models were constructed for these two subsystems. To construct these models, operational states and the telemetry signatures that indicated these states had to be determined. The models were built and compiled, and the correct syntax was verified. Next, it was verified that the expected telemetry signatures were available in the real telemetry stream. Models were edited and recompiled until the model matched the observed behavior.

Displays were constructed using SAMPEX schematics and integrated into the MCS. Tabular alphanumeric displays were generated automatically from a telemetry dictionary. Customization was required to integrate the MCS display into the HP-UNIX Visual User Environment (VUE) environment that enabled the MCS to use the multiple desktop feature of VUE. This platform was not supported at the start of the project.

SAMPEX commanding then had to be analyzed and understood. Inference engine objects were automatically built from a command dictionary. Inference engine rules that translate operator selections into appropriate commands and attribute values were developed. A client process to assemble binary command images based on command attributes as defined in the command dictionary also was developed.

SAMPEX real-time command procedures had to be understood. TPOCC Systems Test and Operations Language (TSTOL) currently is being used for the task. For IMACCS, TSTOL procedures were used as a basis for defining equivalent procedures in the MCS Procedure-Automation-Language-for-Spacecraft syntax. The operator interface and infrastructure to automate pass operations was developed.

Software was developed to capture data from LTIS 550 and make it available to the rest of the MCS application using the MCS software bus. Approximately 500 lines of C++ code were generated to build the interface and some other assorted configuration.

Configuration/setup: The MCS used for IMACCS was initially configured at the vendor’s site. To integrate it with the other IMACCS COTS products, it had to be moved to the SEF where all other COTS resided. This necessitated that the other COTS products (e.g., Talarian’s RTWorks, VI’s Data Views, other public domain utilities) used by the MCS also had to be moved and installed at the SEF prior to moving the MCS application files. This was accomplished with standard UNIX utilities, such as tar, compress, cp, and ln. Other standard UNIX system administration tasks were required, including setting up accounts and granting permission. Altair continued to experiment with installation options by distributing the application across different central processing units (CPUs) and file systems. The tcsh shell scripting language—which is a superset of the language used by the UNIX C-shell, csh—was used to start and stop the system, execute data archival and playback requests, and construct ASCII flat-file databases from the command and telemetry dictionaries. There is no easy way to count script lines of code, especially because a logical line in a script can span multiple physical lines, but approximately 500 logical lines (1 line = 1 UNIX command) is probably close. It took 2 staff-weeks to get the scripts developed, installed, and debugged.

Significant road blocks: There was some difficulty determining data validity within the context of the SAMPEX telemetry stream because data is partitioned into non-time-contiguous blocks.

The short schedule forced Altair to make some design compromises and be rather stingy about which features to implement. One technique that seemed to work well was to define specific objectives for the project early and remain focused on these objectives. It is important not to allow nice-to-have features to creep into the baseline and distract project team members. At least, defer them until the end of the project.

A liaison was used as an interface between the IMACCS team and the MCS vendor. This was important because the liaison could more readily obtain the information needed by the vendor for developing the MCS.

Understanding the intricacies of SAMPEX command formatting was another challenge. Again, it helped to have knowledgeable liaisons who could find answers quickly. Altair’s considerable experience in adapting designs from prior command systems also was helpful.

Vendor: Extremely helpful within the limits permitted by the size of the company. It is easy to overtax the resources available. They tend to think their tool is the most appropriate one for every problem; often times they are correct.

Recommendations: More data is needed to establish the compatibility of Altair's MCS with the needs of a spacecraft integration system. Certainly it is the indicated COTS product for mission operations, particularly where automation is a goal.

7.2 BBN's Probe

Function performed: Trending

General impression: Very good; very easy to do basic trending, more difficult to do bit-level manipulation of data

Good points: Extensive decommutation capabilities. Most mature COTS product used in IMACCS. Most flexible COTS product directly out of the box. Very quick to get plots generated.

Bad points: Need training to be maximally effective in using it. Very picky about having fixed input.

Integration effort: Easy. The output of the MCS had to be clearly defined, a data dictionary for telemetry parameters had to be created, and the integrator had to learn how to use the mathematical analysis component. C code did not have to be written to perform trending. However, 10 lines of C code were written to perform a time routine. A couple hundred lines of Perl scripts had to be written to automatically generate Probe scripts from the raw data files. The Probe scripts automatically generated a data dictionary entry for each telemetry parameter from the raw input data file. This action took a few weeks.

Configuration/setup: Data dictionaries had to be generated for each raw input data file. It was a relatively easy process to simply trend data. It took 1 week to learn how to use Probe and a few days to start trending a sample data file. It took 1 week to learn Perl and a couple of weeks to write the Perl scripts to automate the data dictionary generation process. It took a couple more weeks to learn how to use Probe to mathematically manipulate the data and produce automated trending reports.

Significant road blocks: Learning Probe and Perl. Probe was learned by following a tutorial given to another engineer who had taken a class. Perl was learned by reading manuals and studying examples.

Vendor: Good, but too busy. Responses to technical questions were received within 24 hours. Licensing issues and tape receipt took longer.

Recommendations: Recommend that others use it. However, it works best when input is as structured and fixed as possible. Good for integration tool as well as trending.

7.3 Storm Integration's Precision Orbit Determination System

Function performed: Implements the orbit determination function, i.e., takes tracking data as input and determines a state vector for the satellite

General impression: PODS is a shell built around an implementation of GEODYN, which is a proven system. PODS Version 1.0, which was used in IMACCS, was incomplete in its implementation of the underlying GEODYN functions, but otherwise made the job of orbit determination more intuitive. PODS was easy to use, although certain maintenance functions were more labor-intensive, e.g., updating force model data.

Good points: Makes the orbit determination process much easier and understandable with its graphical user interface and its integration with STK.

Bad points: Error messages originating from the underlying GEODYN need improvement. They do not adequately explain the current problem to the user. PODS had a bug (possibly only in its HP version) that caused it to get lost if the user moved around directories in another window. After returning to the PODS window, PODS would frequently be unable to execute a command because it could not locate the directory it wanted. The program would then have to be restarted. Another bug found was that it would write numbers in meters to a screen hard-labeled in kilometers.

PODS reports were supposed to indicate when the orbit determination process had converged, but would say that it did not converge when it obviously had. Other times it appeared to be diverging, yet a good solution was produced. This problem needs to be corrected.

Between PODS and STK, epochs are entered on several different panels. The exact meaning of each of these and their interdependencies needs to be explained in the manual.

Integration effort: On the input side, raw tracking data was written by the FEP to a file that RDProc would later read, process, and write to another file. PODS would read the files output by RDProc. The state vector output by PODS was automatically fed into STK. Because PODS is a high-fidelity orbit propagator, an ephemeris could be generated at the same time. The ephemeris was written to the same STK vehicle file format used by the rest of STK.

Configuration/setup: PODS setup with current force model data (solar flux, geomagnetic index, polar motion, timing coefficients) is a lengthy process. The vendor supplied files to help in this process, but the data was 2 years old and essentially useless. Current force model data should be shipped with the product. For the IMACCS demonstration, the extra accuracy provided by having current force model data was not needed. Therefore, a complete setup was not needed. However, a sufficient portion of the setup was completed to gain an understanding of how much work was required to do the complete setup. Even after the data is available, it needs to be formatted such that PODS can read it. This is a tedious text-editing task.

The geodetic locations of the ground tracking facilities and certain spacecraft parameters (e.g., area, mass, drag coefficient) must be specified for PODS. Once specified, data entry is quick.

Significant road blocks: PODS Version 1.0 does not allow the user to filter out measurement types. For IMACCS, range data needed to be filtered out. Although not documented, it was found that range data could be filtered if the user interface was bypassed. However, this had to be repeated for each run. It was decided that it was easier to modify the RDProc code and filter range data with RDProc. PODS Version 1.1 will provide this capability.

Vendor: Provided sufficient help

Recommendations: Make sure sufficient tracking data is available. Because of the problem with PODS reporting whether the orbit determination had in fact converged, it is not easy to tell when enough tracking data had been used.

7.4 Analytical Graphics' Satellite Tool Kit

Function performed: Acquisition data generation: ground station contact times; azimuth, elevation, and range data; lighting; node crossings; and passage over the South Atlantic anomaly (SAA) and another region of interest [imprecisely yclept electron contamination region 2 (ECR2)]. It also was used to generate ephemeris data for other applications.

General impression: A good product. It could be made more useful for the purpose at hand by reducing interactive requirements.

Good points: The basic STK package, including Chains, is easy to learn and use.

Bad points: For the purpose at hand—routine product generation—too much interaction is required.

The PL is expensive and difficult to learn and use. In fairness, however, it should be pointed out that the PL was not essential to this project, and efforts to use it were abandoned after approximately 1 week's effort, not full-time. It is likely that this difficulty would be overcome with more effort and some training. If so, the PL would tremendously enhance STK's versatility and potential for customization.

Input/output (I/O) options for orbit propagation are too restrictive. For example, the ephemerides saved in the vehicle files are always in Earth-centered rotating coordinates, and the high-precision orbit propagator requires ECI J2000 vectors as input.

Integration effort: Little integration effort was required. STK was used basically as a standalone tool. Its main products were on paper. ASCII files containing ephemeris data were provided to other applications (MatLab and the vector coalignment utility). Those applications provided the interface.

Configuration/setup: Setup required specifying the location of ground stations (latitude, longitude, elevation) and the SAA and ECR2 (polygons with vertices specified by latitude and longitude points). Chains was used to define a constellation comprising all the ground stations to produce a time-ordered list of potential access times. This was very easy, except for problems in computing passage over the polygonal regions (described in the following paragraphs). In addition, the exact parameters output by Chains had to be specified interactively each time the program was run.

Significant road blocks: STK provides the option to define area targets—polygonal regions on the Earth. There is no way specifically to request times when the spacecraft is passing over these regions. (Asked for access times, STK computes line-of-sight access.) This was easily overcome by defining a nadir-pointing, narrow field-of-view sensor on the spacecraft and by computing times when that sensor viewed the region of interest.

More serious, however, was a problem in STK. In some instances, it computed the SAA and access times erroneously. The problem manifested itself in three ways:

1. It computed some anomalous zero-duration access times. This could not be worked around. In actual support, this could be overcome by manually editing output or writing a script to do so.
2. More seriously, the program sometimes produced access times 180 degrees around the world from the actual target. This did not happen with the SAA. Though not needed, it was discovered that the problem could be overcome by specifying a maximum range from the region's centroid. This would not work for all possible regions or all possible spacecraft.
3. The solution for ECR2 was related to the solution to the third problem. ECR2 is approximately a band around the center of the Earth. STK was unable to compute fly-over times for this region when defined directly by the latitude and longitude points used operationally by the FDF. This was overcome by tiling the region, i.e., breaking it up into six smaller overlapping polygonal regions and using Chains to treat it as a single region. This required significant trial and error because STK sometimes computed access accurately and sometimes not, with no apparent pattern. In the end, however, a workable tiling pattern was developed.

Another problem arose with station masking. Although STK provides this capability, it did not work in all cases, but in others it worked flawlessly. There was no obvious characteristic to distinguish masks that worked from those that did not. There is no good workaround for this. In this case, a simple minimum elevation was used. In actual operations, this would probably be acceptable for some missions and unacceptable for others.

Vendor: Generally very helpful. Easily accessible, although often it was not possible to talk directly with technical experts. In a few cases, the person handling the call failed to report back on the status of the investigation, necessitating follow-up calls. The team discovered that dealing with the STK field representative was far more effective**.**

Recommendations: Close contact with the vendor is essential. This ensures the availability of the proper expertise to tackle problems and also helps to influence the vendor to incorporate desirable features in subsequent releases of the product. It was discovered that MatLab could far more easily accomplish some tasks that were being attempted using STK.

The product adequately performed its intended function. In addition, the IMACCS team has provided suggestions to the vendor for enhancements that would further increase its usefulness. The team also is investigating ways to provide batch execution capability through the use of other COTS tools.

7.5 CSC’s Raw Data Processor

Function performed: Converts tracking data formats. It takes raw 72-byte tracking data as input and converts it to GEOS-C format, which is an ASCII format that PODS can read.

General impression: RDProc more or less did its job—converting tracking data formats. RDProc was easy to use once it had been modified and rebuilt. It required only changing two file names in a file for each run.

Good points: Easily extensible to other cases and data types.

Bad points: No documentation

Integration effort: None

Configuration/setup: Minimal

Significant road blocks: RDProc was modified to provide the capability to filter out range data, a deficiency encountered with PODS. Problems were encountered when RDProc was reassembling after this activity because the documentation did not adequately explain which files were needed. Source code provided with RDProc was mixed with other files that did not belong to RDProc.

RDProc used a table that translated facility identifiers into a set of unique identifiers. It was determined that this capability was not needed for IMACCS and it had to be disabled.

RDProc had not been tested with Doppler data and was in fact not processing it correctly due to an ambiguous specification. This problem was identified and code changes to RDProc were made to fix the problem during IMACCS.

Vendor: Not applicable

Recommendations: RDProc is a CSC-developed software product. Its documentation is incomplete and deficient. If subsequent activities will use the software, a complete set of good documents is needed because modifications most likely will be required.

7.6 The Math Works' MatLab

Function performed: Mathematical algorithm implementation and visual image generation to integrate attitude determination into IMACCS, format orbit product (IIRVs and EPVs) for transmission, and compute SAMPEX beta angles

General impressions: MatLab was easy to use after a short time. It had relatively simple hooks to graphic functions. Not all options were tried.

Good points: MatLab had excellent vector and matrix algorithms, several useful built-in functions, a simple plotting package, and a relatively simple user interface package with graphics functions.

Bad points: For large applications, MatLab programs resemble FORTRAN or C programs in many ways, especially file I/O. For example, many of the file I/O functions described in the users guide tell the user to reference a C programmers manual for a better description of format statements. For some reason, disk I/O was also very slow. A bug was uncovered when subtracting two numbers that were equal to each other. Many times the difference was not zero but some very small number, which caused problems. The vendor explained that they were still using 16-bit arithmetic because they were having difficulties implementing 32-bit arithmetic.

Integration effort: Relatively easy. It had to read STK-generated ephemeris files and RTWorks RTplayback files. This required a fair amount of formatting code. C programming was not required to integrate MatLab into IMACCS, but MatLab does provide the capability to access C or FORTRAN code as an executable if needed.

Configuration/setup: A lot of installation was required. Several FDF FORTRAN subroutines had to be imported and converted to obtain the final product, an effort that took 1 to 2 months.

Significant road blocks: Sixteen-bit arithmetic, and having to debug converted algorithms done by someone who left in the middle of the project

Impressions of vendor: Both helpful and relatively easy to reach

Recommendations: An excellent product for mathematical modeling, the team definitely would recommend it to others. Although MatLab is considered a COTS product, it is less of a COTS product than many others. It is more like a programming language and requires a considerable amount of configuration effort to get it to perform attitude determination.

7.7 Loral Test and Information Systems Model 550

Function performed: IMACCS used an LTIS 550 communication FEP as its interface between the HP workstations and the Nascom serial communication lines. This chassis was connected to the Nascom lines in one building and controlled by a Sun workstation located in the IMACCS room in another building. Connectivity between the front end and the Sun and HP workstations was provided by a campus Ethernet LAN.

Data to be received by the IMACCS LTIS 550 front end consisted of Nascom 4800-bit blocks containing primarily two different types of data:

1. Tracking data (72-byte tracking records)
2. Spacecraft telemetry (CCSDS packet telemetry frames and packets)

Also, the Nascom blocks came from two different types of NASA tracking stations:

1. Wallops Island—4800-bit blocks in modified Digital Data Processing System (DDPS) format
2. Deep Space Network (DSN) sites—4800-bit blocks in DSN-GSFC interface block (DGIB) format

This meant that the beginning of the blocks were similar, but the data portion of the blocks started and ended at different locations.

General impressions: A very powerful and highly configurable front end, it is probably overkill for SAMPEX. Its use required some study; fortunately, a wealth of documentation exists.

Good points: Fast, highly configurable

Bad points: Expensive, and because all functions are performed in hardware, it is difficult to distribute (should that become desirable).

Integration effort: Integration with Nascom was simple. Integration with downstream functions was more challenging. LTIS 550 requires and provides a particular output structure that is both a blessing and a curse. The tag-and-data structure allows data to be tapped at any point along its stream; it also makes troubleshooting a bit harder than otherwise. Loral provides an application programming interface structure in the form of gathers. Gather libraries can be built and reused.

Configuration/setup: After connecting the front end to the serial communication lines from Nascom (RS-422 clock and data), the front end was configured to begin receiving blocks using the point-and-click interface provided by LTIS. This interface also was used to configure filters to identify the different types of blocks by both data type (tracking, telemetry) and data format (DDPS, DGIB). This provided an immediate capability for sorting out the Nascom blocks and counting the number of blocks. Once data blocks were identified, they could be directed to the proper processes for further processing.

Tracking Data Processing

To process tracking data, a modified LTIS algorithm was developed to filter blocks by searching for a byte pattern at a user-specified offset and then passing on a specified string of bytes. This was used to select only SAMPEX tracking data out of all tracking data being received and then pass on the 72-byte tracking record. The filter and extraction offsets were modified to accommodate the different locations of the data inside DDPS blocks and DGIBs. The 72-byte records were passed through a standard bit-reversal algorithm and were then ready for collection by the HP workstation.

A data-gather process was created in C code on the HP workstation to use the LTIS frame gather to collect tracking data records on the HP. This process was built around a sample data gather program provided by LTIS and consists of about 170 lines of code. The process executes continuously on the remote HP workstation to collect any SAMPEX tracking records and write them to a file. These records are then processed in a batch mode whenever desired.

Tracking data was ready to be processed about 2 weeks after the LTIS chassis was installed.

Telemetry Data Processing

Spacecraft telemetry is passed from ground stations to control centers as an unsynchronized bitstream in 4800-bit blocks. Processing telemetry data required extracting the data portion from telemetry blocks and sending it through a hardware synchronizer to derive CCSDS frames. The LTIS Nascom block algorithm library supported extracting the data portion of a DGIB, but a modified version of that algorithm had to be developed to extract the data portion of a DDPS block. This data was then looped back through the serial interface card, and CCSDS frames were extracted by configuring the card to search for the CCSDS frame synchronization pattern. This provided CCSDS frames for input to CCSDS processing algorithms.

LTIS algorithms were used to process the cyclic redundancy code (CRC) on the frames, sort the frames by virtual channels (VCs) 0, 1, 2 and 7, and extract applications process identifiers (APIDs) 1 through 45 from the frames. Point-and-click operations also were used to log VC1 data onto a small computer system interface (SCSI) disk inside the LTIS 550 chassis at 900 kbps for later replay at lower rates.

The team initially attempted to configure the front end using standard algorithms to extract all 3000 data fields from the packets and perform engineering unit conversion as needed. This was done using Perl scripts to scan the SAMPEX project database and generate an ASCII configuration file for LTIS 550. This was then merged with the configuration information created using the point-and-click interface to produce the full configuration.

However, using the standard algorithms required using a packet_extract algorithm for each of the 3000 parameters. This algorithm took a significant amount of memory and required the front end CPU to copy an entire packet into a separate buffer for each of the 3000 parameters. It was impossible to load a configuration with all 3000 parameters in the front end due to memory limitations, and performance was very poor.

To solve this problem, LTIS provided sample C code for a decommutation algorithm, along with the corresponding header file that defined the format of the fields to extract for one of the 35 APIDs. Another Perl script was developed to use the SAMPEX project database and generate the proper header files for each of the 35 packet types. The header files consisted of the parameter tag value, starting byte location in the packet, number of bytes, and whether the output bytes should be in forward or reverse order. This resulted in 35 new algorithms whose code was totally generated by a Perl script reading the SAMPEX project database.

As the Perl script produced each field definition for the decommutation process, it also created ASCII configuration definitions to be imported into the LTIS 550 configuration. All parameters required parameter definitions for the front end to correspond to the fields extracted by the C code routines. For simple byte or multibyte extracted fields, these definitions just defined the parameter name and type (e.g., character, integer, float). Other parameters required further processing on LTIS 550 such as bit field extraction or engineering unit conversion. For these, the proper LTIS 550 processing routines were configured and associated with the extracted field definitions just created. Along with creating extraction code and configuration definitions, the names of the processed telemetry parameters also were written to a file for use by the data-gather process described later.

These algorithms were then compiled and installed in the front end, and a full configuration was developed by merging the generated configuration with the point-and-click configuration information. This configuration fit easily into front-end memory and did not consume excess CPU resources.

The total lines of glueware code to configure LTIS 550 for full SAMPEX telemetry decommutation and engineering unit conversion can be broken down into two categories:

1. Perl scripts and code developed and tested by IMACCS personnel, which required weeks of effort

2. C code, LTIS 550 ASCII configuration definitions, and current value table definitions that were generated by the Perl scripts (This code was generated in approximately 15 seconds by running the scripts.)

Table 7–1 shows the total lines of code.

The final step in receiving telemetry required developing a data-gather process to run on a HP workstation to collect parameter values and pass them into Altair's MCS. This was done by taking a sample current-value-table gather routine from LTIS and merging it with code from Altair. The gather process calls LTIS library routines to collect the current value of each of the 3000 front-end parameters once each second and then calls Altair routines to pass the collected values into the MCS. This process required about 260 lines of code. This completed the flow of telemetry parameters through the front end and into the MCS for analysis and display.

Command Transmission

To support commanding, the main problem was determining the exact process and format required to send commands to the spacecraft using the CCSDS Command Operating Procedure (COP) Protocol. Once this was done, it was decided that the easiest way to implement this would be to build the entire Nascom block on the HP workstation and then use LTIS 550 as the interface to send the block to the Nascom message switch system. The actual transmission of the block to the front end used the inverse of a data gather—a muxwrite—to write the 600 bytes of the block to the multiplexer bus of the front end. This was done using standard LTIS library calls. The front end then verified that the block contained Renaissance source code, calculated a CRC for the block, and sent it to the Mission Support System.

Table 7–1. Total Lines of Code

Developed Perl scripts and C code	
Perl script to merge analog and discrete parameter information from seven project database files into single file	620
C code template (.c file) for packet decommutation	187
Include file template (.h file) for decommutation routine	102
Perl script to read project database file and produce .c and .h file for each of 35 packet types, corresponding ASCII configuration file for LTIS 550, and current value table parameter list	1,490
Computer-generated C code, configuration definitions, and current-value-table definitions	
Total lines of .c files (35 APIDs * 187 lines/template)	6,545
Total lines of .h files (35 APIDs * 102 lines/template) plus packet fields	5,925
Total lines in ASCII definition file • LTIS 550 ASCII parameter definitions = 3988 • Total file size = 673,116 bytes	16,625
Current-value-table parameters	3,083
LTIS 550 decommutation glueware summary	

Total lines of code developed and tested	2,110
Total lines of code generated by scripts	32,178

One Final Problem

One final problem encountered was a first-in-first-out (FIFO) overflow error on the front end that always occurred about 2 minutes into a 900-kbps pass from Wallops Island just before the playback dumps occurred. A FIFO overflow on the front end can be the result of trying to push too much data through the device or it can also be an indication of a code error in some algorithm running on the front end. The main problem is that there are no good ways to see exactly what the problem is on the remote front end. Also, when the problem occurs, communication between the control workstation and the front end degrades in various ways.

Solving this problem required using the SCSI disk logging capabilities on the front end to log some Wallops Island passes and collect test data. This data was then replayed to establish that the problem was repeatable and to determine exactly which data was being processed at the time of the problem. The problem was not rate related and turned out to be garbage packets from the spacecraft. The solution was to put more checks in the LTIS packet processing code to discard data with improper packet lengths and resume on the next frame.

After the final code fix from LTIS, the front end processed all data flows properly with approximately 18 to 20 percent CPU utilization during the 900-kbps data flows from Wallops Island. Utilization was approximately 0 to 1 percent during 16-kbps data flows from DSN sites.

Significant roadblocks: Several times, various boards would cease functioning; the symptoms were not always obvious. Also, memory usage was very inefficient, but the vendor corrected this problem.

Vendor: Extremely generous with their support, providing engineering support and replacement hardware quickly and often. This was good because the product did not function exactly as advertised when it arrived.

Recommendations:

1. Move to a consistent usage of a standard protocol for all data transmission across NASA.

The first major lesson came in understanding the wide variety of formats used within NASA to transport various types of data. The exact same type of data is carried by different format Nascom blocks, depending on whether the data is coming from the Wallops Island tracking station or a DSN site. The Nascom block itself contains no field that indicates what format it is. The only way to determine the format is to determine which type of data (tracking or telemetry) is contained in the block and to use that information along with the source code field to deduce the block format. This scheme works for SAMPEX, but another mission could have a different scheme. However, even for SAMPEX, the team discovered undocumented deviations in the Nascom blocks.

Destination and format codes are checked to try to determine the type of data. This process is complicated by the fact that one destination code can cause data to be sent to up to eight sites, and the team needed to figure out a bit mask to use in filtering the data wanted.

After processing tracking data from both Wallops and DSN sites and telemetry data from DSN sites, the team thought it had all the data formats figured out and just needed to work on the 900-kbps Wallops telemetry. However, the team could not seem to get consistent performance with Wallops data. Some frames and packets were being received, but the frame and packet counts did not match the Payload Operations Control Center (POCC) counts.

The problem turned out to be an extra, undocumented 6-byte time field after the Nascom header and before the data field. These extra 6 bytes did not exist on Wallops tracking data. They only appear on 900-kbps data and do not follow the documented DDPS standard. After changing the offset and length parameters used to extract data from the Wallops circuit switched blocks, CCSDS data processing worked fine.

Also, the format for sending commands to the spacecraft is very complex. Once the proper command string has been built, it must be converted to CCSDS format and then packed into the proper format Nascom blocks, depending on which site is being used for commanding.

The main lesson learned here was that the detailed communication formats used by a satellite are not documented in one place and may not be correct in the documents found anyway. Many different formats are used by different NASA organizations to send the same data. This adds to confusion and complexity of receiving spacecraft data, along with the limited availability of Nascom 4800-bit block serial interfaces.

Moving to a consistent usage of a standard protocol such as IP for all data transmission across NASA would eliminate the need for all of the custom front ends used today to deal with the variety of formats. With IP and standard LANs, any workstation could easily send and receive data with formatting and decommutation done in software. Today, specialized hardware front ends are needed primarily to deal with the bitstream synchronization issues associated with Nascom blocks and CCSDS frames.

2. Do not use a current-value-table type of gather to collect parameters from the front end.

The other main issue with IMACCS relates to the decision to perform parameter extraction and engineering unit conversion on the front end. These parameters (with an associated flag) are stored in a current value table on the front end and are passed to the data-gather process running on the HP workstations once per second. All 3000 parameters are passed once per second regardless of whether any of them have changed or how many times they might have changed. In the case of SAMPEX, extracting all parameters on the front end increases the amount of data to be passed to the HP workstation by a factor of about 10 times. During a SAMPEX 16-kbps data pass, the data rate between the front end and the workstation was approximately 180 kbps over the Ethernet LAN. This is not an unmanageable rate, but the function could be done more efficiently.

The current-value-table approach becomes more of a problem when processing recorded housekeeping (VC1) data. In this mode, data collected across 24 hours is processed and stored in an archive for later extraction to generate trending reports. The current-value-table approach that collects a data sample once each second does not lend itself to running at accelerated rates. If the recorded data is replayed too quickly, then rapidly changing values may be overwritten in the table before it is sent. If the table is sent too frequently, the Altair system may not be able to keep up with the changes.

These problems could be helped by using a different data gather type and perhaps moving some of the parameter decommutation work to the HP workstation. The Altair software currently maintains its own current value table, which reflects the state of the spacecraft at a given time. When a table is passed in from the data-gather process, the Altair software examines its entries and moves those that have changed into its own table. If the data process collected a parameter from the front end each time it changed (rather than a table with 3000 changed and unchanged entries once every second), then the Altair current value table could be updated more often with less work. The current LTIS would support such a configuration. To conserve network bandwidth, telemetry decommutation and engineering unit conversion could be moved from the front end to the HP workstation.

However, these data-gather issues are due to the fact that a separate front-end system is needed to deal with the complex communication formats. If that problem is resolved with standard communication protocols that deliver data directly to a workstation, the data-gathering problems are greatly reduced.

Data flow architecture, particularly the tag-and-data format, provides a highly flexible front-end data mechanism that should prove useful in an integration and test environment. It allows the user to choose data at any point along its path as it undergoes successive processing. It comes with some problems, however. Nascom and CCSDS frame error information is difficult to associate with the final converted values.

Section 8. Conclusions

IMACCS is considered an overwhelming success. The COTS approach has been substantially established for future GSFC missions with projects taking a hard look at using this approach. Trying this approach for SAMPEX presents a clearer understanding of what is involved in building systems using COTS. In addition, team members are more knowledgeable about the products selected, both their capabilities and shortcomings, and their suitability for satellite ground operations support. In many cases, the team was significantly impressed with the capabilities that the product provided and learned that many products exist that offer the best prospect for automating operations for future missions. In addition, through evaluation, the team determined that available COTS products can be satisfactorily integrated with products developed at GSFC. Another significant finding was that when costing the total life cycle, COTS products offer substantial savings when compared to more traditional methods of developing ground systems.

The team plans to extend IMACCS to handle currently unmet requirements and to further investigate some of the options available with COTS products for increasing automation. In addition, future work is expected to include conducting additional prototype evaluations where a broader set of requirements is implemented with alternative COTS products. Also, more work is needed to determine the best way to estimate the cost of developing a system with COTS products. Plans are to pick a future GSFC mission and use a COTS-based ground system to fly with it.

An empowered team appears to be a viable implementation approach for projects similar to IMACCS. The team was highly motivated to succeed, the timeframe covered by the project was short, and the objectives were well understood. These factors contributed to the success of the project. The factors leading to the success of the process were empowerment, well-defined objectives, accurate reporting, rapid identification and response to problems, and, most importantly, trust in the technical team by the management staff and trust in the management team by the technical staff.

Finally, IMACCS has been a catalyst for reengineering both process and technology for the development of future ground systems within MO&DSD at GSFC.

Appendix A. Requirements Comparison

A.1 IMACCS Requirements Analysis for SAMPEX SRR Summary

Table A–1 illustrates the system requirements review (SRR) section containing the requirements listed, the system operations requirements document (SORD) section, and whether the overall function is implemented in IMACCS. The Applicability column summarizes the information included in Table A–2.

A.2 Requirements Detailed Analysis

Table A–2 summarizes SAMPEX requirements as specified in the SAMPEX SRR for each requirement category. The table illustrates how many requirements were listed in the SORD and how many were relevant to IMACCS, and lists the kind of configuration used for implementation. Of the requirements relevant in IMACCS, some were implemented using one or more of the following: a fifth-, fourth-, or third-generation language, or legacy software code. Analytical Graphics' STK and Altair's MCS are examples of a fifth-generation language. The Math Works' MatLab, which is more similar to a language than a tool, is considered to be a fourth-generation language. A third-generation language is a conventional programming language such as C. Glueware was implemented using a third-generation language. The table also illustrates the number of requirements relevant to SAMPEX that were not implemented in IMACCS.

Table A–3 provides a detailed listing of SRR requirements by category, whether IMACCS has met the requirement, what kind of configuration was used to meet the requirement, and which COTS product was used to implement the function.

Table A–1. SAMPEX SRR and IMACCS Requirements Analysis Summary (1 of 2)

SRR Section	SORD Section	Included in IMACCS	Applicability
1.0 Project Overview	1000	No	Project-level requirements are satisfied by details in later sections.
2.0 Ground Network	2000, but not 2600	No	Ground stations and networks are out of IMACCS boundaries, but many of these requirements can be satisfied reliably with IP-based delivery such as Transmission Control Protocol (TCP)/IP or ISIS and network management using Simple Network Management Protocol (SNMP)-based products.
3.0 Flight Dynamics	3200	Yes	Thirteen of the total 20 requirements are relevant to IMACCS boundaries. Of these, only one is not satisfied. In addition, 13 make use of fifth-generation language products, 4 required fourth-generation language scripts, and none required third-generation language coding (verify glueware included).
4.0 Control Center	3300	Yes	Eighty-five of the total 92 requirements are relevant to the IMACCS boundaries. Of these, 15 are not satisfied. In addition, all 70 make use of fifth-generation language products, 4 required fourth-generation language scripts, and 4 required third-generation language coding.
5.0 Level-Zero Science	3400	No	LZP of science data is out of IMACCS boundaries, but most of these requirements could be satisfied with a suitable COTS front end, the MCS or equivalent, a DBMS, and IP-based delivery such as FTP.
6.0 Command Management	3500	Yes	Twenty-seven requirements are relevant to IMACCS boundaries. Of these, all are satisfied with legacy software.
7.0 Test and Simulation	2600, 3600	No	The test and simulation requirements are out of IMACCS boundaries. Recent test and simulation systems are already largely COTS-based, and other COTS products are suitable.

Table A–1. SAMPEX SRR and IMACCS Requirements Analysis Summary (2 of 2)

SRR Section	SORD Section	Included in IMACCS	Applicability
8.0 Communications	4000	No	The communications requirements are out of IMACCS boundaries, but these are already satisfied by mostly COTS-based mechanisms and would benefit further from IP-based standards.
9.0 Special Requirements	3700	No	Maintenance of flight software is satisfied by operational procedures using COTS CASE tools. Image storage and compares are facilitated by COTS tools. Multimission support is primarily operational.

Table A–2. Requirements Analysis with Configuration Summary

SRR Section	Requirements Category	SORD Rqmts	Relevant Rqmts	5th GL	4th GL	3rd GL	Not Satisfied
3.0 Flight Dynamics	Orbit Determination	4	4	4	2	0	0
	Attitude Determination	3	3	3	4	0	0
	Mission Analysis	7	0	0	0	0	0
	Planning Aids	2	2	2	0	0	0
	Special	4	4	2	0	1	1
3.0 Flight Dynamics Total		20	13	11	5	1	1
4.0 Control Center	Functional Interfaces	13	9	8	0	1	1
	Real-Time Telemetry	15	13	12	0	2	1
	Special Real-Time Telemetry	7	7	6	1	0	1
	Real-Time Telemetry Report	10	10	6	0	0	4
	Playback	5	5	4	1	0	1
	Commanding	11	11	11	0	1	0
	Display	12	12	12	0	0	0
	Systems Test and Operations Language (STOL) (User Procedure)	5	4	3	1	0	1

	History	8	8	7	1	0	1
	Database	2	2	1	0	0	1
	Scheduling	4	4	0	0	0	4
4.0 Control Center Total		92	85	70	4	4	15
Grand Total		112	98	81	9	5	16

Acronyms

APID	applications process identifier
ASCII	American Standard Code for Information Interchange
CCSDS	Consultative Committee for Space Data Systems
CERES	Center for Research Support (U.S. Air Force)
CIGSS	CSC Integrated Ground Support System
CLCW	command link control word
CMS	Command Management System
COP	Command Operating Procedure (Protocol)
COTS	commercial off-the-shelf
CPU	central processing unit
CRC	cyclic redundancy code
CSC	Computer Sciences Corporation
DBMS	database management system
DDF	Data Distribution Facility
DDPS	Digital Data Processing System
DGIB	DSN-GSFC interface block
DSN	Deep Space Network
ECR2	electron contamination region 2
EPV	extended precision vector
ETU	engineering test unit
FDF	Flight Dynamics Facility
FEP	front-end processor
FIFO	first-in-first-out
FOT	Flight Operations Team
FTP	File Transfer Protocol

GDS	ground data system
GOTS	Government off-the-shelf
GREAS	Generic Resource, Event, and Activity Scheduler
GSFC	Goddard Space Flight Center
HILT	Heavy Ion Large Telescope
HP	Hewlett-Packard
IIRV	improved interrange vector
IMACCS	Integrated Monitoring, Analysis, and Control COTS System
Inmarsat	International Maritime Satellite
I/O	input/output
IP	Internet Protocol
kbps	kilobits per second
LAN	local area network
LEICA	Low-Energy Ion Composer Analyzer
LTIS	Loral Test and Integration System
LZP	level-zero processing
MCS	Mission Control System
MIDEX	Medium Explorer
MO&DSD	Mission Operations and Data Systems Directorate
NASA	National Aeronautics and Space Administration
Nascom	NASA Communications
NDI	nondeveloped item
Pacor	Packet Processor
PC	personal computer
PL	Programmer’s Library
POCC	Payload Operations Control Center
PODS	Precision Orbit Determination System

RAM	reusable application module
RDProc	Raw Data Processor
Renaissance	Reusable Network Architecture for Interoperable Space Science, Analysis, Navigation, and Control Environments
SAA	South Atlantic anomaly
SAMPEX	Solar Anomalous and Magnetospheric Particle Explorer
SCP	spacecraft command processor
SCSI	small computer system interface
SEAS	Systems, Engineering, and Analysis Support
SEF	Systems Engineering Facility
SMEX	Small Explorer
SNMP	Simple Network Management Protocol
SORD	system operations requirements document
SRR	system requirements review
STK	Satellite Tool Kit
STOL	Systems Test and Operations Language
TCP	Transmission Control Protocol
TPOCC	Transportable Payload Operations Control Center
TSTOL	TPOCC Systems Test and Operations Language
UMSOC	University of Maryland Science Operations Center
VC	virtual channel
VME	Versa-Module European
VO	Visualization Option
VUE	Visual User Environment

Bibliography

Bracken, M. R., et al., “IMACCS: An Operational COTS-Based Ground Support System Proof-of-Concept Project,” *First International Symposium on Reducing the Cost of Spacecraft Ground Systems and Operations*, Chilton, Oxfordshire, UK, September 1995

NASA/GSFC, *MO&DSD Process Guide Proposed by Renaissance* (draft version 1.9), Greenbelt, Maryland, August 1995

Monfort, Lt. Col. Ralph, Center for Research Support, “A New Acquisition Management Philosophy for COTS-Based TT&C Systems,” *National Security Industrial Association Symposium*, Sunnyvale, California, August 1995

NASA/GSFC, Code 500 Renaissance Team, *Renaissance Data/Process Flows Presentation.* Greenbelt, Maryland, June 1994

Pendley, R. D., E. J. Scheidker, D. S. Levitt, C. R. Myers, and R. D. Werking, “Integration of a Satellite Ground Support System Based on Analysis of the Satellite Ground Support Domain,” *Third International Symposium on Space Mission Operations and Ground Data Systems*, Greenbelt, Maryland, November 1994

Stottlemyer, A. R., A. Jaworski, and S. R. Costa. (October 1993). New Approaches to NASA Ground Data Systems, *Proceedings of the Forty-fourth International Astronautical Congress, Q.4.404*, Graz, Austria, October 1993

Werking, R. D., and D. R. Kulp, “Developing the CSC Integrated Ground Support System,” Poster Presented at the Seventh Annual AIAA/Utah State University on Small Satellites,. Logan, Utah, September 1993

Wheal, C.A., *Application of State Space Modeling Techniques to Satellite Operations,* Altair Aerospace Corporation, Bowie, Maryland, 1993

Table A–3. Detailed Listing of SRR Requirements by Category

ategory	Requirement	IMACCS	Implementation
	1.0 Project Overview (SORD 1000, 1100, 1200, 1300)	N/A	Mission Requirements
	Low-cost, multimission with common spacecraft elements for quick response to science community (3 years start to launch), with average mission cost of $30 million (fiscal year 1988)		
	Spacecraft has 24-hour autonomy and data storage with 12 hours storage of science data, onboard attitude determination and control, stable spacecraft clock with adjust, onboard limit checking and safehold logic, CCSDS, and primarily single string design		
	GSFC institutional ground segment, ground station only, 24 hours per day, 7 days per week POCC, command management with use of onboard spacecraft control processor (SCP)		
	Spacecraft—mechanical (solar array), thermal (onboard temperature regulation with passive thermal control and thermostatically controlled heaters), power (direct energy transfer, solar array, battery, shunt, autonomous operation and fail checks), attitude control (3-axis, zenith pointing instruments, Sun-pointing panels, magnetometer, momentum wheel, torquer rods, onboard attitude determination with software control and analog safehold), data system (Small Explorer Data System telemetry, command, radio frequency and timing, and instruments: HILT, LEICA, mass spectrometer telescope, and proton/electron telescope)		
	2.0 Tracking and Data Acquisition (SORD 2000, but not 2600)	N/A	Network and Ground Statio Requirements
mmary	Launch and early orbit full coverage (7 to 11 passes per day)		
	Wallops, 2+ passes per day (command, telemetry, tracking)		
	DSN, 2+ passes per day (command, telemetry, tracking)		
	Prepass data flows (test commands, simulated telemetry)		
	Commanding (throughput, echo, DGIB)		
	Telemetry (real-time throughput, DGIB, contingency VC0)		
	Tracking (two-way Doppler, ranging tones: CTV testing, coherent tracking simultaneous with command and telemetry)		
	Data archive (postpass playback, 72+ hours archive)		
ecial	Spacecraft clock correction support (measure ground equipment delay, capture universal time code of command link transmission unit start)		
	Data accountability statistics (number of blocks, transfer frames, and quality transfer frames by VC)		SNMP
	Ground system management (fault isolation and coordination)		SNMP
	Ground system scheduling (generic, special operations requirement, available 24/7, critical support, and schedules: strawman, forecast, 7-day, updates)		GREAS
	3.0 Flight Dynamics Facility (SORD 3200)		
bit etermina- n	Orbit vector prediction (onboard orbit propagator, UMSOC orbit determination)	4GL, 5GL	STK, MatLab (EPV script)
	Atmospheric drag coefficients (onboard orbit propagator, UMSOC orbit determination)	4GL, 5GL	MatLab (Note: coefficients are computed by GSFC, but just packaged for delivery.)
	Ground track prediction (antenna field of view, pole crossings, radiation zone crossings)	5GL	STK
	Orbit predict data (verify against onboard orbit propagator)	5GL	STK (high-precision orbit

Category	Requirement	IMACCS	Implementation
			propagator), PODS, LTIS 550
itude termina- n	Attitude determination (verify onboard attitude determination and control, backup for onboard attitude determination)	4GL, 5GL	MatLab (Attcal script), LTIS 55 MCS, STK
	Magnetometer bias determination (backup for onboard)	4GL, 5GL	MatLab (Attcal script), LTIS 55 MCS, STK
	Monitor attitude sensors	4GL, 5GL	Probe (data preparation and tre scripts), LTIS 550
anning Aids	Ground station coverage predictions	5GL	STK
	Eclipse predictions	5GL	STK
ecial	Slant range data prediction	No	
	Beta angle predictions	5GL	STK
	Sun and magnetic field coalignment predictions	3GL Legacy	Rehosted SAMPEX FORTRAN program (Note: could have use MatLab script.)
	SAA crossing prediction	5GL	STK
ssion alysis	Network profile	N/A	Premission analysis studies ca generally be performed using tools such as STK, MatLab, an Probe with Altair simulated dat but some are unique to missio parameters.
	Sun and magnetic field vector coalignment	N/A	
	Eclipse profile	N/A	
	Beta angle profile	N/A	
	Orbit decay study	N/A	
	Orbit determination study	N/A	
	Other mission analysis studies as requested	N/A	
	4.0 Payload Operations Control Center (SORD 3300)		
nctional erfaces	Nascom, ground stations, spacecraft	5GL	LTIS 550
	Ground stations scheduling element	No	Used SAMPEX POCC scheduling, but could be done with GREAS.
	FDF: Attitude data	5GL	LTIS 550, MCS
	FDF: Planning products	5GL	MCS, could be enhanced with DBMS.
	CMS: Loads, tables, images	3GL, 5GL	LTIS 550 and MCS interface w SAMPEX CMS.
	Pacor: Quality statistics and archive data	N/A	LZP not targeted with IMACCS but could be done with many fr ends and data reduction such a the MCS.
	FOT: operator interface	5GL	MCS
	Project PC: telemetry for offline analysis	5GL	MCS, FTP
	Project PC to UMSOC: E-mail	5GL	SMTP/MIME
	Nascom test and simulation equipment: command and telemetry	N/A	
	Project database	5GL	MCS
	Launch site video	N/A	
	Flight software maintenance	N/A	
al-Time lemetry	Receive real-time actual and simulated data	5GL, 3GL	LTIS 550, MCS, Altair Simulat
	Provide real-time Nascom block, transfer frame, and packet-level	5GL	LTIS 550, MCS

ategory	Requirement	IMACCS	Implementation
	lock status and quality statistics		
	Strip out CLCW and provide to telemetry processing, and use in COP-1 command verification and retransmission	5GL, 3GL	LTIS 550, MCS, glueware
	Unpack nonscience VC0 packets using project database unpack parameters	5GL	MCS
	Calibration and conversion of [state, engineering (integer, real, float, polynomial project database), raw] data	5GL	LTIS 550, MCS
	Display and print tabular and graphic representations of telemetry, status, accounting, event, and configuration data	5GL	MCS
	Perform limit checking for red and yellow high and low limits and project database-defined limit values on option and allow temporary limit changes	5GL	MCS
	Configuration monitoring automatically of 20 critical parameters and 5 configuration files	5GL	MCS
	Strip attitude (attitude control subsystem) packets as requested and ship to FDF	5GL	MCS
	Spacecraft images: collect, store, and report images and provide uplink verification	N/A	Interface with SAMPEX CMS.
	Interval data collection, archival and download to project PC (accessible by UMSOC)	No	COTS approach would be to bu reformatting glueware and FTP products.
	Generate, store, and display event messages (time tagged, descriptive messages for telemetry, command, system and operator significant events)	5GL	MCS
	Record and archive for 30 days all real-time and recorder dump data, commands, event messages, and operator or procedure input	5GL	MCS
	Record and archive for mission lifetime current and previous flight and ground software, including project database	N/A	Maintenance of flight software an operational requirements, bu would be supported with standa library and DBMS tools.
	Operator/STOL access to real-time telemetry for conditional and logical expressions and data values displays	5GL	MCS
ecial Real-ne Telem.	Delog, display, and report onboard spacecraft event log messages	5GL	MCS (still being configured afte 90 days)
	Provide snapshot of spacecraft power system operation, including computation and display of five derived power system parameters	4GL, 5GL	Probe using Perl scripts that prepare data and generate Pro scripts to automate trending fro actual data ranges.
	Correct and display spacecraft clock values from PB5J to calendar format	5GL	LTIS 550 (still being configured after 90 days)
	Collect and display history of key instrument parameters from previous orbit	No	Not included in IMACCS, but could be done using LTIS 550 and MCS on VC2 data.
	Display packet time in human-readable format by identifier for diagnosis	5GL	LTIS 550 (still being configured after 90 days)
	Display and STOL access summary of packet types flags	5GL	MCS
	Display and STOL access of instrument rate data (expansion of compressed data and calibration)	5GL	MCS
al-Time lemetry ports	Status and accounting report	5GL	LTIS 550, MCS
	Event message log	5GL	MCS

:ategory	Requirement	IMACCS	Implementation
	Limits summary report	5GL	MCS
	Configuration monitor report	No	Not included, but could be done with MCS.
	CLCW report	5GL	Manual report, data available ir MCS.
	Packet reception report	5GL	Manual report, data available ir MCS.
	Downlink image report	No	Not included, but could be done with LTIS 550 and UNIX difference program.
	Spacecraft event log report	5GL	MCS
	Instrument history report	No	Not included, but could do with LTIS 550 and MCS.
	Interval data archive report	No	Not included, but could do with LTIS 550 and MCS.
ored lemetry ayback	Receive and archive VC1 recorder dumps, provide status and accounting information in real time	5GL	LTIS 550, MCS
	Strip out attitude data for FDF	5GL	MCS
	Engineering and attitude partitions only one partition at a time	5GL	MCS
	Sequential print	5GL, 4GL	MCS and Probe (with scripts)
	Special packed-data processing (?)	No	
ommanding	Rapid callup of project database-defined commands and CMS-provided loads	5GL	MCS
	Command buffer, mode, and sequence control	5GL	MCS
	Transmit commands to ground station	5GL	LTIS 550, MCS
	Formats (CCSDS, project database-defined packets, subfields, dynamic from sequence number and control command bits, Nascom blocks)	5GL	LTIS 550, MCS
	Commanding modes (one- or two-step, verification/CLCW, retransmission enable/disable, COP-1 bypass enable/disable)	5GL, 3GL	MCS and glueware
	CCSDS control commands (unlock, set next expected frame to 0)	5GL	MCS
	Spacecraft user commands: mnemonic, variable subfields and mandatory checks, project database validation with mask, critical command handling	5GL	Manual checks using data available in MCS.
	Transmit commands at 3 seconds per block with status reporting	5GL	MCS
	Table, image, and SCP loads (CMS-provided, call up by file, rapid uplink-delayed verification, image verification)	5GL	MCS
	Command archive (transmitted blocks, command event messages, command echo blocks, operator input)	5GL	LTIS 550, MCS
	Performance (call up within 3 seconds, transmit within 1 second, display event message and status within 2 seconds, verify CLCW within 2 seconds, provide image comparison results within 10 seconds)	5GL	MCS
splay	General (cathode ray tube and hardcopy, STOL message area, standard header, event message area, display names and numbers, user friendly, 2-second update for event driven, 4-second for telemetry driven)	5GL	MCS
	Command displays (status, buffer, uplink image comparison, command mode, load directory)	5GL	MCS
	Telemetry displays (100 project database-defined with flexible format)	5GL	MCS
	Data status and accounting	5GL	MCS

ategory	Requirement	IMACCS	Implementation
	Event message	5GL	MCS
	Out of limits	5GL	MCS
	Configuration monitor	5GL	LTIS 550
	TPOCC (equivalent) configuration and status	5GL	LTIS 550, MCS
	Procedure status	5GL	MCS
	XY plot	5GL	MCS
	Command panel	5GL	MCS
	Functional diagram program display	5GL	MCS
OL	Control all POCC functions through standard STOL user interface language with error and syntax checking, abbreviations and optional prompt mode, and system access	5GL, 4GL	MCS PALS script capability
	Provide direct procedure transfer with integration and test system (same STOL)	N/A	Operational requirement
	Access to telemetry values	5GL	MCS
	Automated procedure capability with standard procedure controls	5GL	MCS
	Broadcast capability	No	
story	Offline processing of archived data (telemetry not science, commands, event messages)	5GL	LTIS 550
	Select by time, rate, directory	5GL, 4GL	LTIS 550 produces history files manual file management or DBMS tools
	Provide all real-time capabilities	5GL	LTIS 550
	Raw dump (Nascom blocks)	No	
	Raw dump (transfer frames)	5GL	LTIS 550 for transfer frames ar packets, IMACCS does not sto Nascom blocks.
	Raw dump (packets)	5GL	LTIS 550 for transfer frames ar packets, IMACCS does not sto Nascom blocks.
	Command and echo delog at block and frame level	5GL	MCS
	Event message delog	5GL	MCS
tabase	Ingest project database (command specifications, telemetry specifications, display definitions)	5GL, 3GL	LTIS 550, MCS, glueware
	Validate and report database	No	Not included, but would use DBMS.
heduling	24/7 scheduling service	No	POCC resource scheduling not included in IMACCS, but would use COTS tool such as GREA
	FOT schedule requests	No	
	Coordinate with SMEX lead scheduling element	No	
	Report schedules	No	
	5.0 Pacor (SORD 3400)	N/A	LZP was not targeted with th IMACCS prototype, but most the requirements could be satisfied with a COTS front er and the MCS.
ut	Synchronize on Nascom blocks (VC0, VC1, VC2) with data quality checks, but process flagged data		
	Data quality statistics (counts, flags, sequencing, CRC)		
ltilevel	Packet format delivery, sorted by APID		
	Annotate missing or incomplete packets		
	Remove redundancy saving best quality		

ategory	Requirement	IMACCS	Implementation
	Delivery to UMSOC daily (Monday through Friday) with 24 hours of data per data set		
strument eckout	Support early turnon and checkout of instruments and forward data to instrument ground checkout system		
rmal erations	Real-time and recorded data, electronic TCP/IP x.25 delivery over 56-kbps Nascom line to UMSOC		
uicklook	Quicklook data to UMSOC for special science requests, instrument malfunction, or data processing unit anomaly		
chive	Archive processed data for 30 days		
	Raw data and Nascom block level for 2 years		
	Reprocess block-level stored data as requested		
	6.0 Command Management (SORD 3500)		The IMACCS prototype reuse the SAMPEX CMS for comma management processing; cou have used a DBMS and the MC
proach	Frequently changing tables and contingency tables and images, including early error detection, continuity for dual SCP system	N/A	
e Setup	Activity files, project database, constants, triggers, events, load input, orbit data with syntax, error, project database checking	Legacy	
ad eneration	Multiple loads (version, days, destinations) for operations, patches, RTS-only loads	Legacy	
	Error and constraint checking for onboard system limits and sequence/mode	Legacy	
	RTP management for four relative time processors	Legacy	
	RTS table management with sequence assignment	Legacy	
	Load editing (manual)	Legacy	
	Send loads to POCC for uplink	Legacy	
ss Plan	Accept user input, format using event file and load information, print, archive	Legacy	
ble aintenance	Multiple versions, two destinations, with edit, report, error checking	Legacy	
plink Table	Ingest binary images	Legacy	
	Store tables and images in partial and composite form	Legacy	
	Store multiple versions for two destinations	Legacy	
	Raw value report	Legacy	
	Images to POCC for uplink	Legacy	
ent Log	Hard copy and archive of chronological event log	Legacy	
splay	Files, load generation process, events with graphics and hardcopy	Legacy	
eports	Operations load and memory table maps	Legacy	
	RTP utilization	Legacy	
	Load generation errors	Legacy	
	File listings and directories	Legacy	
	Pass plans and pass plan generation	Legacy	
	Event logs	Legacy	
rformance	Patch load in 15 minutes	Legacy	
	Display requests in 1 second, report requests in 2 seconds	Legacy	
	Transfer load, table, or image to POCC in 5 minutes	Legacy	
	Install new database in 30 minutes	Legacy	
	Command management 24 hours per day, 7 days per week	Legacy	
	7.0 Test and Simulations (SORD 2600, 3600)	N/A	The test and simulations requirements were not targete for IMACCS; many are suitab

:ategory	Requirement	IMACCS	Implementation
			for COTS solutions, includin legacy approaches.
rtable nulation	Process spacecraft commands and provide telemetry dynamics		
	Receive, store and dump command packets		
	COP-1 simulations and NES values		
	Output telemetry streams		
	Telemetry stream errors		
	Generate and use telemetry value sets		
nulation erations nter	Emulate Wallops and DSN		
	Command and data interfaces to PSS		
	Flow taped spacecraft or PSS data over Nascom		
	Generate Nascom block errors		
ogramma- Formatter	Nascom blocking, deblocking of all command and data rates		
	Nascom source and destination codes for Wallops, DSN		
	I/O status indicators		
	Status and accounting		
	View, archive, and print PDF pass through data		
	Resettable time code generator for block time tags		
mpatibility st Van	Radio frequency compatibility testing		
ta aluation boratory	Generate telemetry tapes for data flows by Wallops, DSN, and Simulation Operations Center		
	8.0 Communications (SORD 4000)	N/A	
	Prelaunch and launch phase		IP, ISIS, TCP/IP
	Mission phase		IP, ISIS, TCP/IP
	Experiment data distribution (Pacor to instrument GSE, UMSOC)		FTP
	9.0 Special (SORD 3700)	N/A	Operational Requirements
ght ftware	Maintain SMEX data system flight software after orbital verification		
	Support flight software integration and test		
ltimission	Plan for common FOT for two additional SMEX missions		
	Concurrent operations and future mission test and simulation support		
	Scaleable design with mission unique labeling on all products and displays		